PROGRAMMING THE APPROXIMATION PROBLEM OF NETWORK SYNTHESIS

FREDERIC E. BECK, JR.

PROGRAMMING THE APPROXIMATION PROBLEM OF NETWORK SYNTHESIS

* * * * *

Frederic E. Beck, Jr.

PROGRAMMING THE APPROXIMATION PROBLEM OF NETWORK SYNTHESIS

by

Frederic E. Beck, Jr.
Lieutenant, United States Navy

Submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE
IN
ENGINEERING ELECTRONICS

United States Naval Postgraduate School
Monterey, California

1 9 5 8

PROGRAMMING THE APPROXIMATION PROBLEM OF NETWORK SYNTHESIS

by

Frederic E. Beck, Jr.

This work is accepted as fulfilling
the thesis requirements for the degree of
MASTER OF SCIENCE
IN
ENGINEERING ELECTRONICS

from the
United States Naval Postgraduate School

ABSTRACT

The basic outline of a digital computer (CRC-102A) program for the numerical solution of the approximation problem in the time domain is presented. The method used, in effect, is a real time convolution and Dirichlet series representation of the Laplace transform to effect a time-to-frequency transformation. The major portions of the digital computer program and the complete flow chart to solve the problem are included.

The composition of a basic library of programs for the solution of all network synthesis problems and the integration of this program into such a library is discussed.

The program for the solution of the roots of an $n^{\underline{th}}$ degree polynominal and the Dirichlet series expansion program were completed during the author's industrial tour at the National Cash Register Company, Electronics Division, Hawthorne, California. I take this opportunity to thank Mr. Henry Kent and Mr. Philip Sunday of that organization for their help in the writing and the debugging of the programs completed during my industrial tour.

Special thanks are also due to Professor E. J. Stewart for his recommendations and constructive criticism with regard to the programming of the overall problem, and to Professors M. L. Cotton and J. B. Turner, Jr., for their guidance and help throughout the preparation of this paper.

TABLE OF CONTENTS

Section	Title	Page
1.	Introduction	1
2.	Statement of the Problem	3
3.	The Approximation Problem in the Time Domain	4
4.	The Network Synthesis Library	25
5.	Conclusions	28
6.	Bibliography	29

LIST OF ILLUSTRATIONS

Figure		Page
1.	A Trivial Case	5
2.	Derivation of h(t)	8
3.	A Triangular Inpulse Response	11
4.	Demonstration Problem	19
5a.	Three - Pole Approximation to h(t) -- Program Solution	22
5b.	Four - Pole Approximation to h(t) -- Program Solution	23
6.	The Basic Network Synthesis Library	27

1. Introduction

The problem of network synthesis is a difficult and complex one. Consider the synthesis process to be divided into the two separate sub-problems of approximation and realization. There are many techniques for the attainment of the first goal, Z(p), the transfer function.[1] Similarly, having attained Z(p), there are many procedures available to realize Z(p) in any one of several different circuit configurations. From the point of view of the electronics research engineer, a tool which would rapidly and accurately compute the circuit components of a network to provide a specified response, at a specified impedance level, in a specified circuit configuration would free the researcher from the tedium of designing the network and also from the temptation of accepting a "second-best" solution. The designer would feel no compulsion to terminate a problem rather than face another "messy" synthesis problem. The modern high-speed digital computer offers the engineer such a design tool, if an appropriate library of synthesis programs is available.

Possibly the greatest advantage the computer avails its users is that once a certain type of problem has been solved, the method of solution may be permanently recorded on punched cards or on magnetic tape or paper tape. The engineer now need only recall that such a problem has been programmed and refer to the catalogue of computer programs rather than to a textbook to refresh his mind on how to solve the problem by paper and pencil.

[1] For a comprehensive summary of approximation methods, the reader is referred to Trans, IRE, Prof Grp on Circuit Theory, Vol. CT-1, No. 3, of Sep 1954. In addition to the summary, the first paper presented by Stanley Winkler, has a definitive bibliography of 240 items on network synthesis.

It is the purpose of this paper to present a program for the solution of the approximation problem in the time domain and to show how this program may be integrated into a library of programs which use conventional methods for the solution of the more common network synthesis problems.

2. Statement of the Problem

The approximation problem as considered in this paper is felt to be defined as:

Given the excitation available or expected at the input terminals of a passive four-terminal network, and the response or output required, or the impulse response, where these parameters are expressed in graphical, or general analytic form; find a rational function expressed as two polynominals, where one is understood to be the numerator and the other the denominator, which describes the transfer function within the limitations on the allowable deviation of the output or on the number of elements to be used (but not both), specified. Provisions are to be included for compensation of dissipative effects. The polynominals will be expressed either as polynominals or as the quadratic factors of polynominals. The function is to be physically realizable.

3. The Approximation Problem in the Time Domain

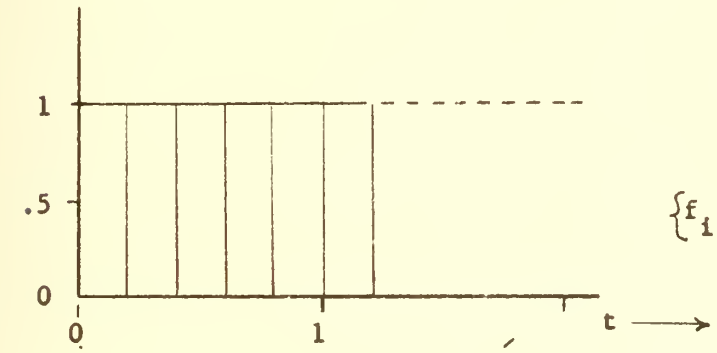
Heretofore, approximation problems in the time domain have involved lengthy and inaccurate frequency domain expansions which do not always lead to realizable system functions. These methods involve complicated integrations, or at best, Taylor series expansions.[1(3)] At any rate, they are not suitable for the flexible program requirements we desire. What we would like to have is a method which would rapidly transform the input-output characteristics of one network into a realizable system function within any desired degree of accuracy, or alternatively, an optimum realization for a specified number of elements. The numerical method of Ba Hli[1(3)] is tailor-made for these requirements. Furthermore, not only will it operate on input-output relationships, but also there is no reason why an impulse response which has been obtained analytically cannot be plotted and introduced into the program at the appropriate point to produce the transfer function for all types of networks, i.e., predictor networks, smoothing filters, interpolation filters, compensation networks and so on. In fact, we may say if a physically realizable h(t) can be defined, we can find the system function. The designer must have a plot or sequence of ordinates which describe the h(t), or the $f_i(t)$ and $f_o(t)$, and use this plot or sequence to enter his problem into the approximation program.

Briefly stated, Ba Hli's method is to find, by a process of synthetic division, a sequence ($\{h\}_A$) which represents the impulse response of the network which in turn relates the input waveform to the desired output;

[1]Freddy Ba Hli, Network Synthesis by Impulse Response for Specified Input and Output in the Time Domain, Tech. Rpt. No. 261, MIT Res. Lab. of Electronics, July 31, 1953 (hereinafter abbreviated as NSIR-BH).

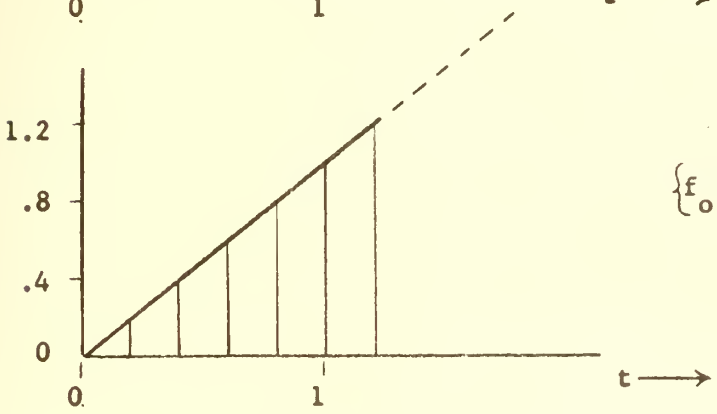
and then to calculate the psuedo-system function H*(s), from $\{h\}_A$ by a simple power series expansion. The H*(s) is a polynominal in s which is the ratio of the $\frac{P^m(s)}{Q^n(s)}$ which is normally considered to be the expression of the system function. In effect, we convert the convolution integral into its component operations and use an iterative substitution method. We shall not reproduce the theoretical reasoning and proof which was so neatly done by Ba Hli, but we shall demonstrate the ease and simplicity of the method with a numerical example.

Before proceeding with a complete example, let us show how the synthetic division process yields the impulse response in a couple of trivial cases, the results of which are well-known. Consider Figure 1,



Unit Step

$\{f_i(t)\}$ = 1, 1, 1, 1, 1, ...



Unit Ramp

$\{f_o(t)\}$ = .2, .4, .6, .8, 1,0, ...

A Trivial Case

Figure 1

the input $f_i(t)$ we take as the unit step whose Laplace transform is $\frac{1}{s}$; $f_o(t)$, the output, is the unit ramp. Its transform, of course, is $\frac{1}{s}2$. If we sample the input and output at intervals of .2 of a time unit, the sampled ordinates for $f_i(t)$ are $\{1,1,1,1,1,1,....\}$, and for $f_o(t)$ $\{.2,.4,.6,.8,1.0,1.4,1.6.....\}$. Now dividing $\{f_o(t)\}/\{f_i(t)\}$ in the manner illustrated:

$$1,1,1,1,1,1..../.2,.4,.6,.8,1.0,1.2,1.4..... \qquad .2,\ .2,\ .2,\ .2,\ .2,.....$$

we have $\{h\}_A = .2,.2,.2,.2,.2,.....$ and if we divide $\{h\}_A$ by Δt, we produce the sequence 1,1,1,1,1,....again the unit step. This, of course, is completely analogous to the process in Laplace transform operations, viz:

$$\mathcal{L}[f_i(t)] \times \mathcal{L}[h(t)] = \mathcal{L}[f_o(t)]$$

or $$H(s) = F_o(s)/F_i(s) \equiv \frac{1/s^2}{1/s} = 1/s$$

We can apply the process in reverse, for example, $\{f_i(t)\} \times \{h\}_A = \{f_o(t)\}$, if we use the unit doublet as $\{h\}_A$ we should differentiate the output. Consider the sequence below which defines a sine wave, with $\Delta t = .1$

$\{f_i\}$ =	.1	.199	.296	.389	.479	.564	.644	.717
$\{h\}_A$ =	10	-10						
	1	1.99	2.96	3.89	4.79	5.64	6.44	7.17
		-1	-1.99	-2.96	-3.89	-4.79	5.64	6.44
$\{f_o\}$	1	.99	.97	.93	.90	.83	.80	.73

$\{f_i\}$ (Cont'd)	.783	.841	.891	.932	.964	.985	.997
	7.83	8.41	8.91	9.32	9.64	9.85	9.97
	7.17	7.83	8.41	8.91	9.32	9.64	9.85
$\{f_o\}$ (Cont'd)	.66	.58	.50	.41	.32	.21	.12

and $\{f_o(t)\}$ is a sequence which defines the cosine which is as expected since the unit doublet acts as a differentiator. Thus we demonstrate the ability of this $\{h\}_A$ to perform as an operator equivalent to H(s) in the frequency domain on functions defined as sequences in the time domain. The transformation of $\{h\}_A$ to H*(s) follows logically from the definition of the convolution integral:

$$f_o(t) = \int_o^t f_i(t)\, h\,(t - \tau)\, d\tau$$

Ba Hli [3] has shown that H*(s) can be represented by the Dirichlet power series,

$$H^*(s) = \sum_{n=1}^{\infty} a_n \exp(-\tau_n s)$$

where the a_n are the areas in our $\{h\}_A$ sequence, and the τ_n are defined by

$$\tau_n = n\Delta t - \frac{\Delta t}{2},$$

that is, the τ_n are measured from the origin to the center of the sampling interval under consideration at the moment.

We may approximate H*(s) as closely as we please by taking our sampling points at smaller and smaller intervals.

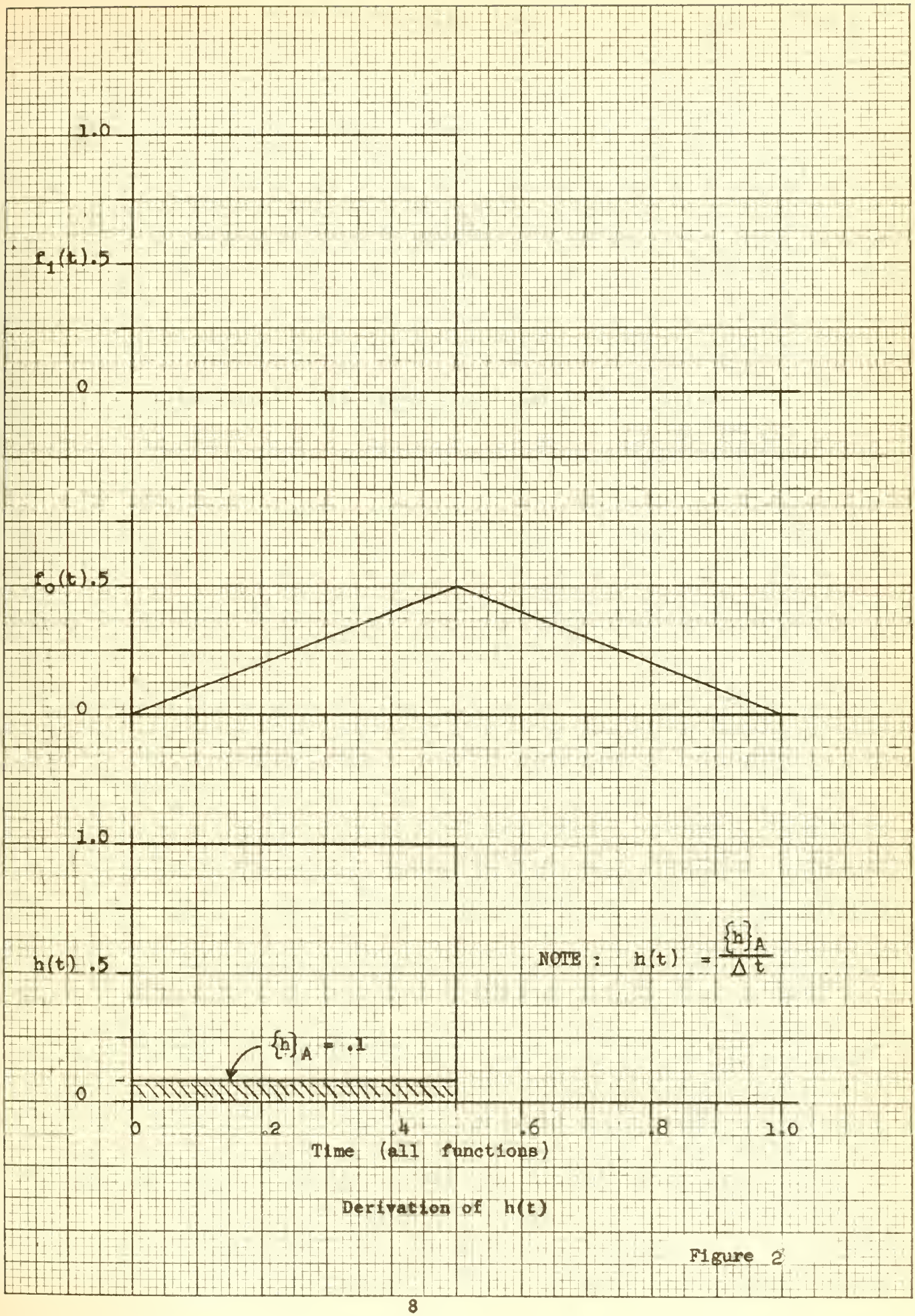
To demonstrate the complete process and provide a basis for discussing some of the more obscure points, let us take another example and follow it through more closely. Figure 2 shows the $f_i(t)$, the desired $f_o(t)$ and the resulting h(t) required, ($\Delta t = 0.1$):

$\{f_i(t)\}$ = 1, 1, 1, 1, 1, 0, 0, 0,

$\{f_o(t)\}$ =.1, .2, .3, .4, .5, .4, .3, .1, 0, 0, 0,

$\{h\}_A$ =.1, .1, .1, .1, .1, 0, 0, 0, 0,

In tabular form, we have



Derivation of h(t)

Figure 2

τ_n	a_n
.05	.1
.15	.1
.25	.1
.35	.1
.45	.1

Substituting into our expression for H*(s), we have

$$H^*(s) = .1\left[1 - .05s + \frac{(.05s)^2}{2!} - \frac{(.05s)^3}{3!} + \frac{(.05s)^4}{4!} - \ldots\right]$$

$$+ .1\left[1 - .15s + \frac{(.15s)^2}{2!} - \frac{(.15s)^3}{3!} + \frac{(.15s)^4}{4!} - \ldots\right]$$

$$+ .1\left[1 - .25s + \frac{(.25s)^2}{2!} - \frac{(.25s)^3}{3!} + \frac{(.25s)^4}{4!} - \ldots\right]$$

$$+ .1\left[1 - .35s + \frac{(.35s)^2}{2!} - \frac{(.35s)^3}{3!} + \frac{(.35s)^4}{4!} - \ldots\right]$$

$$+ .1\left[1 - .45s + \frac{(.45s)^2}{2!} - \frac{(.45s)^3}{3!} + \frac{(.45s)^4}{4!} - \ldots\right]$$

$$= .5 - .125s + .020625s^2 - .00255208s^3 + .0002517968s^4 - \ldots\ldots$$

If we compare this result with that given by Laplace transform methods, we have

$$\mathcal{L}[f_i(t)] = \frac{1}{s}\left[1 - \exp(-1/2s)\right]$$

$$\mathcal{L}[f_o(t)] = \frac{1}{s^2}\left[1 - \exp(-1/2s)\right]^2$$

$$\mathcal{L}[h(t)] = \mathcal{L}[f_o(t)] \Big/ \mathcal{L}[f_i(t)] = \frac{1}{s}\left[1 - \exp(-1/2s)\right]$$

and the power series expansion for $\mathcal{L}[h(t)]$ is

$$.5 - .125s + .020833s^2 - .002604166s^3 + .0002605166s^4$$

which shows the first two terms by Ba Hli's method agree exactly, a 1% error in the third term, 2% in the fourth, and about 4% in the fifth.

Before proceeding let us see how this accuracy may be improved. Clearly increasing the number of terms taken and maintaining the same interval will have no effect on the formation of the second, third, and fourth degree terms whose accuracy we wish to improve and intutively one feels that increasing the number of sample points will; therefore let us increase the number of sample points to, say, 8.

With $f_i(t)$, $f_o(t)$ and $h(t)$ as before we tabulate, for $\Delta t = .0625$:

τ_n	a_n
.03125	.0625
.09375	.0625
.15625	.0625
.21875	.0625
.28125	.0625
.34375	.0625
.40625	.0625
.46875	.0625

$$H(s) = .0625\left[1 - .03125s + \frac{(.03125s)^2}{2!} - \frac{(.03125s)^3}{3!} + \frac{(.03125s)^4}{4!} - \ldots\right]$$

$$.0625\left[1 - .09375s + \frac{(.09375s)^2}{2!} - \frac{(.09375s)^3}{3!} - \frac{(.09375s)^4}{4!} - \ldots\right]$$

and so on; the resulting expression is:

$$H^*(s) = .5 - .125s + .020635719s^2 - .002631105s^3 + .000257032861s^4 - \ldots$$

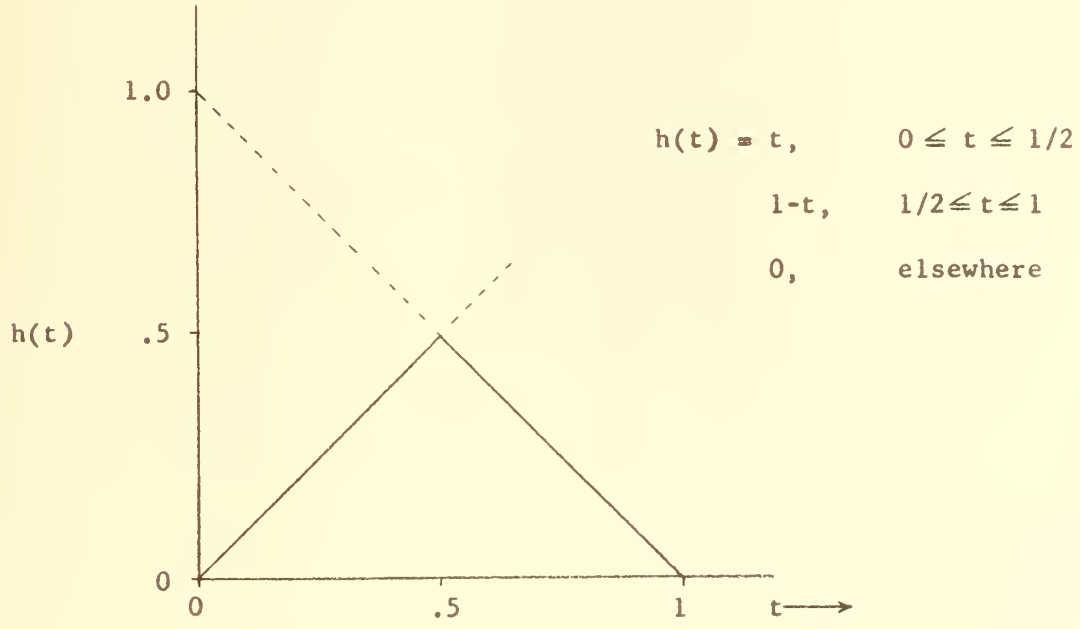
and the new expansion for 8 sample points has produced errors of 0% in the first two terms, .95% in the third term, 1% in the fourth and 1.3% in the fifth. Therefore, the increase in the number of points has obviously resulted in an overall improvement. A further increase to 10 points, produces accuracies of 0%, 0%, .25%, .5%, .8% and 1.25%.

The foregoing results are tabulated in Table 1 for the purpose of

easy comparison.

We recognize that the errors in the higher order terms are caused by replacing the Laplace-Stieljes integral[1] by a truncated series summation. This is equivalent to saying that we have not computed the true "moments" of a_n about the origin.

Consider Figure 3, we have here a triangular impulse response. Let us calculate the second and third moments of this curve and compare the results with the power series expansion of the Laplace transform and also the Dirichlet series expansion. We let b_2 be the second "moment", (it would be more precise to call this by a different term since it is actually $\frac{M_j}{j!}$, where M_j is the moment), b_3 equals the third; thus,



A Triangular Impulse Response

Figure 3

[1] NSIR-BH(3), pg. 15.

TABLE I

True Expansion	.5	$-.125s + .0208333s^2$	$- .002604166s^3$	$+ .0002604166s^4$	$- .000021701388s^5$	$+ \ldots\ldots$
5 Sample Point Approximation	.5	$-.125s + .020625s^2$	$- .00255208s^3$	$+ .0002517968s^4$	$- \ldots\ldots$	
Per Cent Error	0%	0%	-1%	-2.0%	-3.31%	
8 Sample Point Approximation	.5	$-.125s + .020635719s^2$	$- .002631105s^3$	$+ .000257032861s^4$	$- \ldots\ldots$	
Per Cent Error	0%	0%	-.948%	+1.02%	-1.3%	
10 Sample Point Approximation	.5	$-.125s + .02078125s^2$	$- .0025911458s^3$	$+ .0002582503246s^4$	$- .000021431075s^5$	
Per Cent Error	0%	0%	-.250%	-.500%	-.832%	-1.25%

$$b_2 = \frac{1}{2!}\left[\int_0^{1/2} x^3 dx + \int_{1/2}^{1} (1-x)\, x^2 dx\right]$$

$$= \frac{1}{2!}\left[\frac{x^4}{4}\Big|_0^{1/2} + \frac{x^3}{3}\Big|_{1/2}^{1} - \frac{x^4}{4}\Big|_{1/2}^{1}\right]$$

$$= \frac{1}{2}\left(\frac{1}{64} + \frac{1}{3} - \frac{1}{24} - \frac{1}{4} + \frac{1}{64}\right)$$

$$= .03645833$$

$$b_3 = \frac{1}{3!}\left[\int_0^{1/2} x^4 dx + \int_{1/2}^{1} (1-x)\, x^3 dx\right]$$

$$= \frac{1}{3!}\left[\frac{x^5}{5}\Big|_0^{1/2} + \frac{x^4}{4}\Big|_{1/2}^{1} - \frac{x^5}{5}\Big|_{1/2}^{1}\right]$$

$$= \frac{1}{6}\left(\frac{1}{160} + \frac{1}{4} - \frac{1}{64} - \frac{1}{5} + \frac{1}{160}\right)$$

$$= .00781250$$

The trancendental Laplace transform for this function is:

$$H(s) = \frac{1}{s^2}\left[1 - \exp(-1/2s)\right]^2$$

whose power series expansion is

$$H(s) = .250 - 0.1250s + .03645833s^2 - .00781250s^3$$

The Dirichlet series results from

τ_n	a_n
.1	.02
.3	.06
.5	.09
.7	.06
.9	.02

$$H(s) = \sum_{n=1}^{\infty} a_n e^{-\tau_n s}$$

Therefore, considering only the b_2 and b_3 terms, we compute:

$$b_2 = .02 \times \frac{.1^2}{2!} + .06 \times \frac{.3^2}{2!} + .09 \times \frac{.5^2}{2!} + .06 \times \frac{.7^2}{2!} + .02 \times \frac{.9^2}{2!}$$

$$= \frac{1}{2} (.0002 + .0054 + .0225 + .0294 + .0162)$$

$$= .03685$$

Similarly, b_3 is computed as

$b_3 = .008008$

Therefore:

	"Moment"	Laplace	Ba Hli Approx.
b_2	.03645833	.03645830	.036850...
b_3	.00781250	.00781250	.008008...

The obvious method of reducing the error is to take a smaller interval and thereby approach the true integration to a closer degree and since this will be a one time only calculation, we are not adamant about making extra computations to improve the accuracy. We also notice that by making our intervals smaller we reduce those errors which may arise from the process in which we obtain the $\{h\}_A$. For a more complete discussion of the reduction of the inherent error in Ba Hli's method see Appendix II.

Having obtained a psuedo-system function, it yet remains to determine its equivalent as the quotient of two polynominals, the degree of each as yet unknown. We equate our psuedo-function, H*(s) to an expression of the form

$$H^*(s) = \frac{P(s)}{Q(s)} = \frac{p_0 + p_1 s + \cdots p_m s^m}{q_0 + q_1 s + \cdots q_{n-1} s^{n-1} + q_n s^n}$$

and solve for the unknown p's and q's, since these coefficients express a ratio we may arbitrarily set any one of them we choose, (normally q_o)

equal to one without any loss of generality.

We notice that the series of equations which results from this operation will always have the form:

$$P_0 = h_0 q_0$$

$$P_1 = h_1 q_0 + h_0 q_1$$

$$P_2 = h_2 q_0 + h_1 q_1 + h_0 q_2$$

$$0 = h_3 q_0 + h_2 q_1 + h_1 q_2 + h_0 q_3$$

$$0 = h_4 q_0 + h_3 q_1 + h_2 q_2 + h_1 q_3 + h_0 q_4$$

$$0 = h_5 q_0 + h_4 q_1 + h_3 q_2 + h_2 q_3 + h_1 q_4$$

$$0 = h_6 q_0 + h_5 q_1 + h_4 q_2 + h_3 q_3 + h_2 q_4$$

For $m = 2$, $n = 4$; r, the relative degree between P(s) and Q(s), is obviously two. We see that we have seven equations in eight unknowns, which is as it should be for the expression of a ratio, and hence, requiring that we set q_0 (or any other coefficient) equal to some convenient value. The solution of this portion of the problem is considered in detail in Appendix III.

We choose r, the relative degree of numerator and denominator by observing the behavior of h(t) around 0. Since we know from the initial value theorem of Laplace Transform Theory,[1]

$$\lim_{t \to 0} h(t) = \lim_{s \to \infty} sH(s)$$

we may say that the function must behave as

$$\frac{p_m s^m}{q_m s^m}$$ as t approaches zero.

[1] M. F. Gardner and J. L. Barnes[6], page 265.

Thus, if h(t) starts out as a step

$P(s)/Q(s) \longrightarrow 1/s$ as s approaches infinity; or is h(t) starts out as a ramp $P(s)/Q(s) \longrightarrow 1/s^2$ as s approaches infinity.

To program the computer to sense this limit and thus decide whether h(t) is an impulse, step, ramp, or higher order function, we first determine h_0, i.e., h(0). Since in some cases this may result in division by zero, we perform a first-order backward interpolation, using h_1 and h_2 to find h_0. This is approximate but certainly is sufficiently accurate to determine whether or not h(t) starts as a jump function or as a ramp, which is the primary decision in the "selection of relative degree routine" (see Appendix I, page I-1). If it is decided at this point that h(t) is a first-order approximation to a ramp function, we must further decide whether it is truly a ramp or a higher order function, i.e., $1/s^3$, $1/s^4$, etc. These decisions are based on the fact that the $n^{\underline{th}}$ derivative of an n-order polynominal is a constant, the process is flow charted for $+1 \leq r \leq -4$ on page I-2-1 of Appendix I. The routine may be easily extended to encompass all possibilities which may arise.

We know that we can obtain any desired degree of accuracy in the terms of our power series expansion and further that we can correctly determine the relative degree between our numerator and denominator polynominals. We have shown that we can compute the coefficients of our H*(s) and we know that since this is a ratio with arbitrary coefficients we may multiply the expression by the impedance level factor to set any required impedance level. But what of the approximation error, i.e., the error caused by not taking an infinite number of terms to approximate trancendental functions?

[1]Ba Hli has shown that in the case where the Laplace transform of h(t) is rational, it will be recovered exactly. NSIR-BH,(3) pgs. 41-44.

$N = 5 - 1 + 1.5(5) + 2 = 13.5$, which we take as 14.

Therefore, we can write

$$\frac{p_0 + p_1 s + p_2 s^2 + \ldots + p_{12} s^{12} + p_{13} s^{13}}{q_0 + q_1 s + q_2 s^2 + \ldots + q_{13} s^{13} + q_{14} s^{14}} = h_0 + h_1 s + \ldots h_{11} s^{11} + \ldots h_{27} s^{27}$$

this expression leads to 23 equations in 24 unknowns, one of which, customarily q_0, we set equal to 1, without error. To avoid filling sheets of paper with useless symbology, we shall terminate this example here and turn to another pair of functions.

We shall compute, using the program, approximate impedance functions for a pair of input-output time functions and check the computer solution analytically. To do this, we choose the functions illustrated in Figure 4.

$f_i(t)$ is a rectangular pulse

$$f_i(t) = 1 \quad 0 \leq t \leq 1$$
$$0 \quad \text{elsewhere}$$

$f_o(t)$ is an inverted sectioned parabola

$$f_o(t) = .5t^2 \qquad 0 \leq t \leq 1$$
$$= -t^2 + 2.0t - 1.5 \qquad 1 \leq t \leq 2$$
$$= .5\,(t-3)^2 \qquad 2 \leq t \leq 3$$
$$= 0 \qquad \text{elsewhere}$$

for $\Delta t = .1$, the following sequences obtain

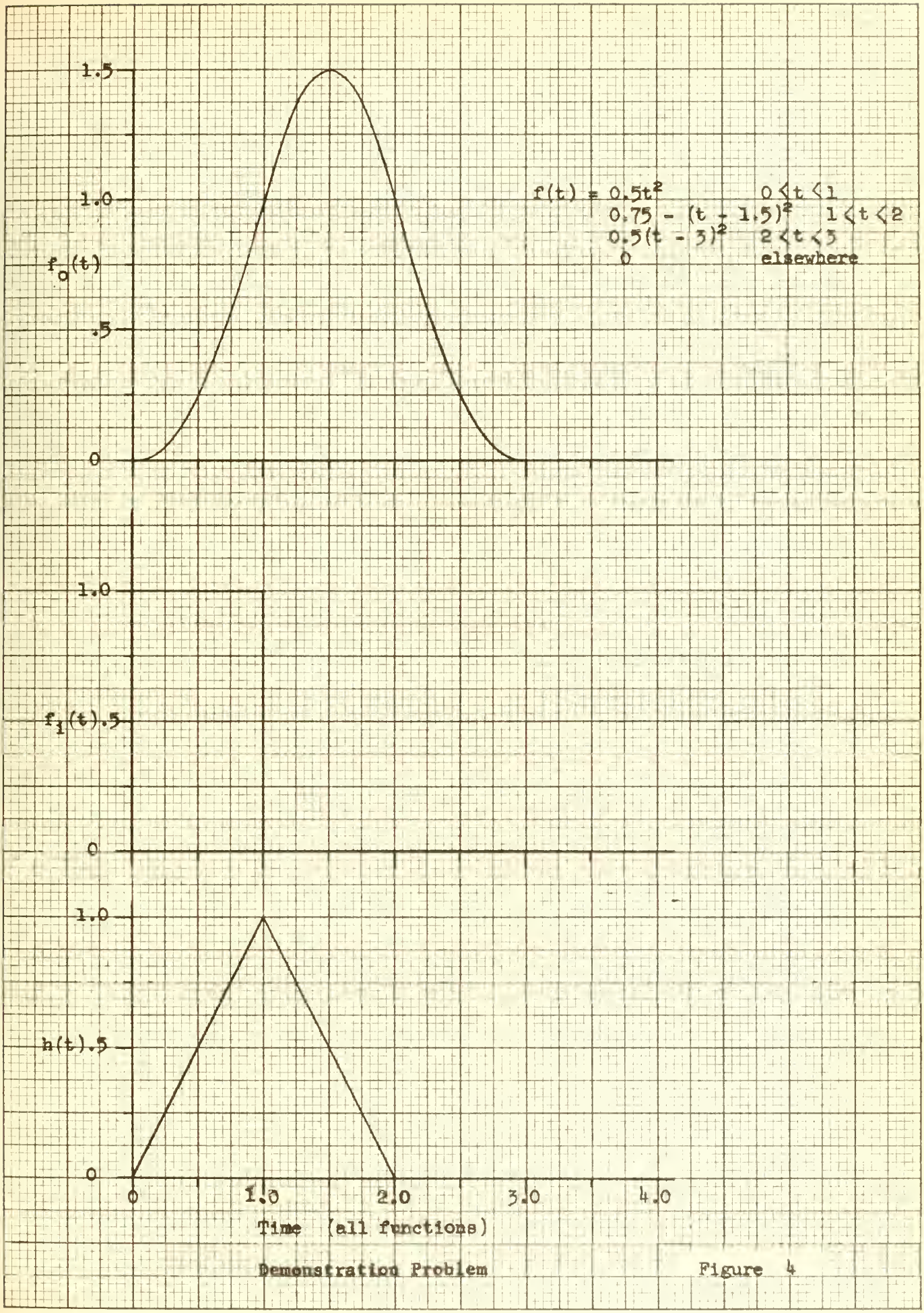
$\{f_i(t)\}$ = 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

$\{f_o(t)\}$ = .005, .020, .045, .080, .125, .180, .245, .320, .405, .500,

.590, .660, .710, .740, .750, .740, .710, .660, .590, .500,

.405, .320, .245, .180, .125, .080, .045, .020, .005, .000,

Synthetic division of $\{f_o(t)/f_i(t)\}$ yields:



Demonstration Problem

Figure 4

$\{h\}_A$ = .005, .015, .025, .035, .045, .055, .065, .075, .085, .095,

.095, .085, .075, .065, .055, .045, .035, .025, .015, .005

which describes h(t) as the triangular function also shown in Figure 4.

We know that the Laplace transform from $f_i(t)$ is:

$$\mathcal{L}\left[f_i(t)\right] = \frac{1}{s}\ (1 - e^{-s})$$

and the Laplace transform for $f_o(t)$ is:[1]

$$\mathcal{L}\left[f_o(t)\right] = \left[\frac{1 - e^{-s}}{s}\right]^3$$

Thus H(s), which should be $\left[f_o(t)\right]$ / $\left[f_i(t)\right]$ or $\left[\frac{1 - e^{-s}}{s}\right]^2$, which is the transform of a triangular pulse[2].

We expect H*(s) as computed by our program to be a good approximation to the power series expansion of $\left[\frac{1 - e^{-s}}{s}\right]^2$.

$$\left[\frac{1 - e^{-s}}{s}\right]^2 = 1.000000000 - 1.000000000s + .583333333s^2$$
$$- .250000000s^3 - .086111111s^4 - .025000000s^5$$
$$+ .006299603s^6 - .001405423s^7 + \ldots.$$

The program approximation with single interpolation, i.e., $\Delta t/2$, produced:

$$H^*(s) = .999999999 - 1.000000975s - .583340832s^2 - .250007019s^3 +$$
$$.086115077s^4 - .025001670s^5 - .006300172s^6 - .001405587s^7 + \ldots$$

If we now solve for the impedance function using n = 3, and r = -2 (since h(t) is a ramp function in the vicinity of zero);

$$Z(p) = \frac{0.999999998 - 0.049971105s}{1.0 + 0.950029868s + 0.366689961s^2 + 0.062506119s^3} \quad \text{or;}$$

$$= \frac{(-.049971105)\ (s - 20.011564640)}{(s + 2.277182620)\ \left[(s + 1.79464120)^2 + 1.950589991^2\right]}$$

[1] Gardner and Barnes[6], Appendix A, No. 6.08.

[2] Gardner and Barnes[6], Appendix A, No. 6.07.

The inverse Laplace transform of this function is:

$$4.4132006227\ e^{-2.27718262t} + 4.465570186\ e^{-1.79464120t} \sin(1.95058999t - 81.217^{\circ})$$

The plot of this inverse superimposed on the desired h(t) is shown in Figure 5a.

If we desire to improve this approximation we go to a 4-pole network. The program computes the system function as:

$$\frac{2.67414555s^2 - 8.1622705s + 135.9105227}{s^4 + 10.59828491s^3 + 51.1404969s^2 + 127.7483847s + 135.9105230}$$

which we expand as:

$$1.0681279249 \left\{ \frac{(s + 23.98432161)}{\left[(s+3.017294581)^2+(.932721456)^2\right]} - \frac{(s + 20.00984647)}{\left[(s+2.281847873)^2+(2.901655110)^2\right]} \right\}$$

and determine the inverse Laplace transform to be:

$$24.010971222\ e^{-3.017294581t} \sin(.932721456t + 2.547^{\circ})$$

$$-6.612687409\ e^{-2.281847873t} \sin(2.901655110t + 9.295^{\circ})$$

This inverse is plotted in Figure 5b. Note the improvement with the addition of the other pole. The approximation can, of course, be further improved by adding still more poles.

Suffice it here to say, that the existing program will solve for the impedance function, given the time-series data describing the input and output waveforms.

Before concluding this section, we estimate the required solution time for the total approximation problem in the time domain.

We assume the following conditions obtain as a basis for our estimate:

1. h(t) is not a closed function, i.e., K = 4 (see Appendix II). If h(t) is closed, that is, if h(t) is completely described in 4 or less time units the running time for the synthetic division routine and for the com-

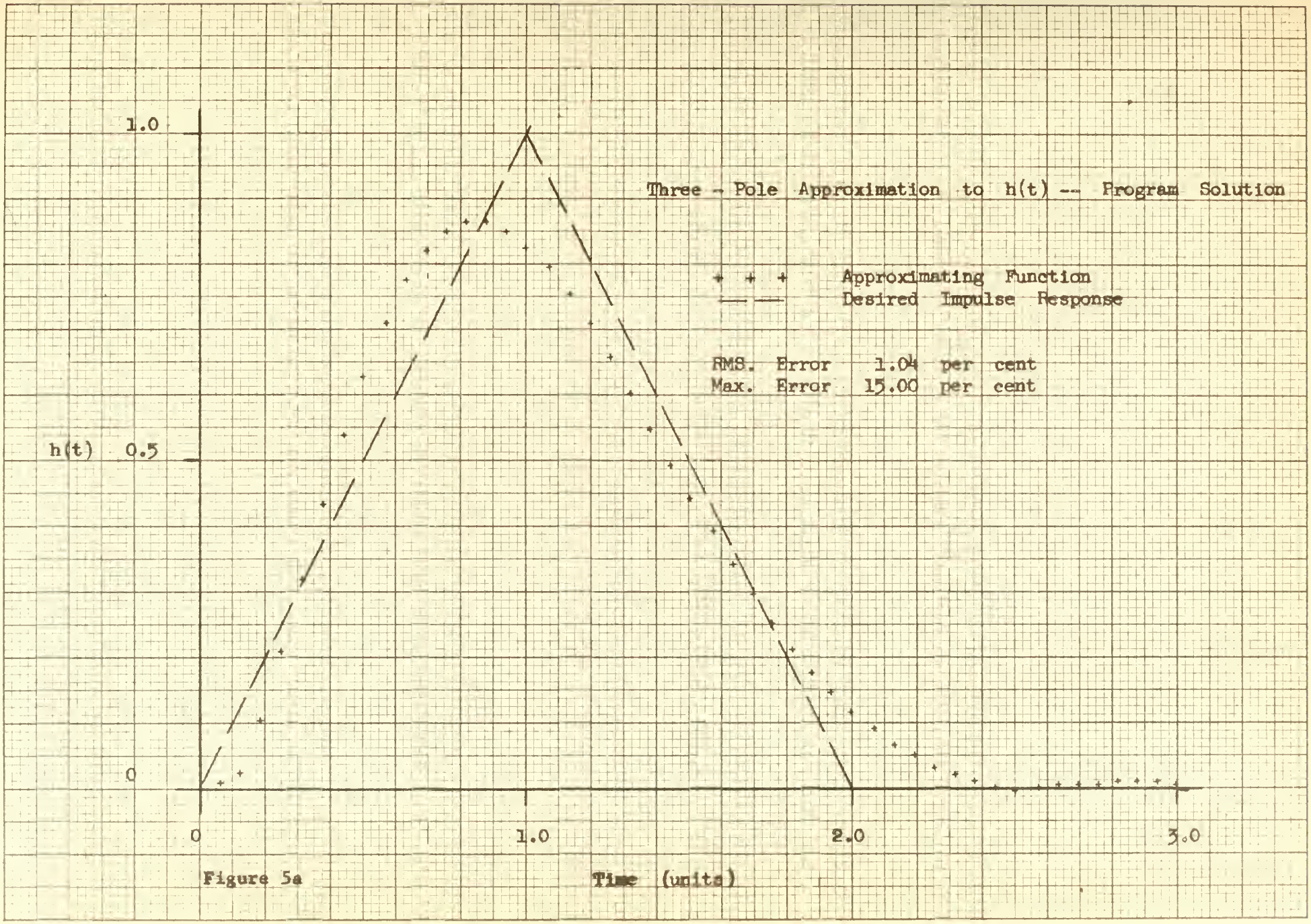

Three - Pole Approximation to h(t) -- Program Solution
+ + + Approximating Function
— — Desired Impulse Response
RMS. Error 1.04 per cent
Max. Error 15.00 per cent
h(t)
1.0
0.5
0
0
1.0
2.0
3.0
Time (units)

Figure 5a

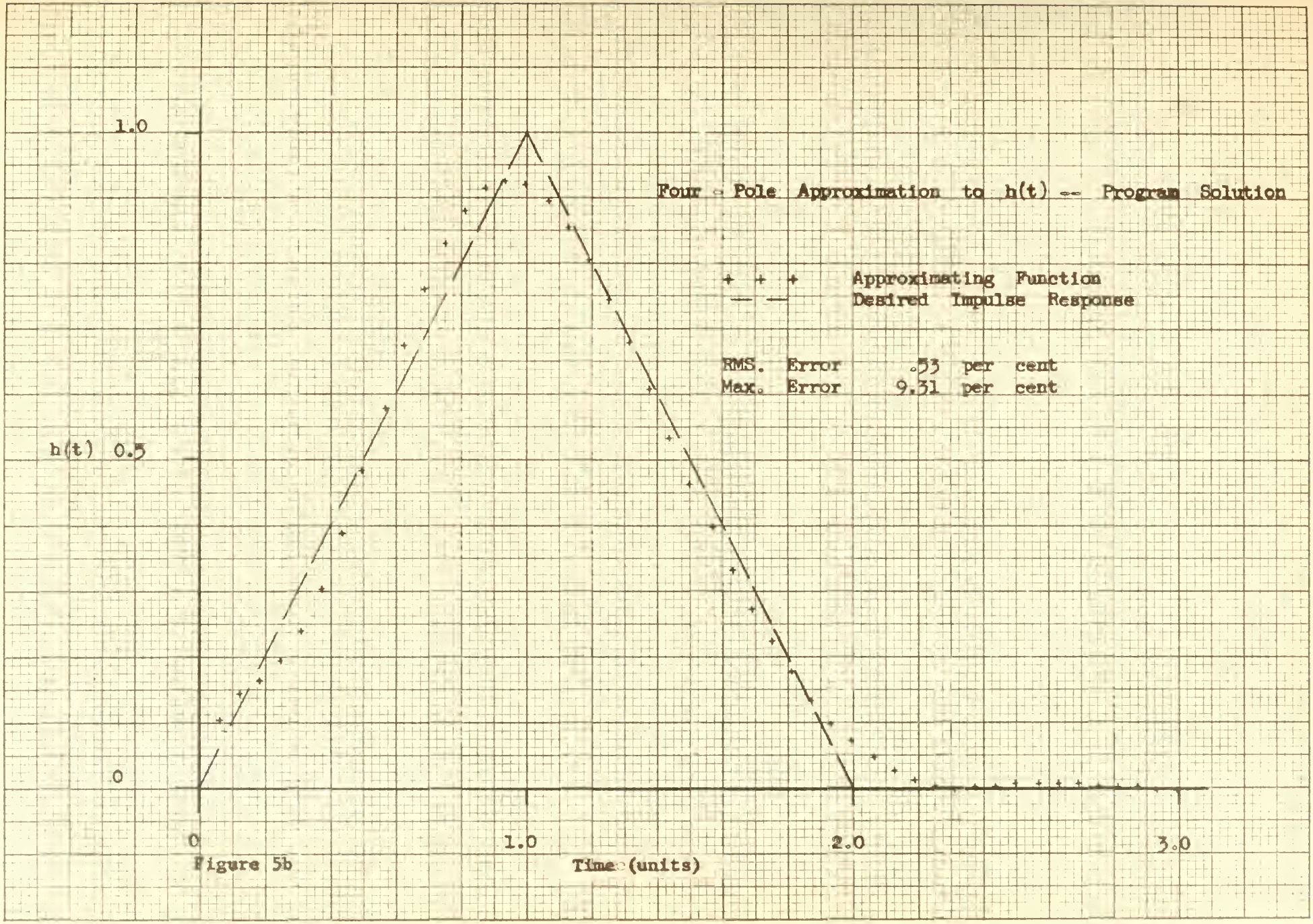

Four - Pole Approximation to h(t) -- Program Solution
+ + + Approximating Function
— — Desired Impulse Response
RMS. Error .53 per cent
Max. Error 9.31 per cent
1.0
h(t) 0.5
0
0
1.0
2.0
3.0
Time (units)

Figure 5b

putation of the psuedo-system function, H*(s), will be proportionately reduced.

2. H(s) starts as a step function, i.e., r = -1. This is the worst case we encounter as regards computation time for the psuedo-system function, since we must compute 2(n + 1) + r terms of the expansion of H*(s) in order to solve the system of simultaneous linear equations.

3. The interpolation decision is such that one intermediate interpolation calculation is performed. This is the intermediate condition, and requires twice the computation time for a no-interpolation decision and one-half the time for the four-point interpolation.

We consider the overall program (for time domain approximation) to be sub-divided into three, (four if Z(p) is desired in factored form or if compensation is to be provided) major computations. Thus we have:

Sub-division	CRC time	Word-times
Synthetic Division to produce $\{h\}_A$	5 min*	72×10^4*
Dirichlet power series expansion H*(s)	n x 3 min	$(n-1) \times 44 \times 10^4$
Matrix Inversion to compute Z'p	$.064\ n^3$ min	$.96\ n^3 \times 10^4$
Factor Z'p to apply loss compensation	(2n-1) x 9.4 min	$(2n-1) \times 144 \times 10^4$

*Estimated, all other values measured.

For a 10-pole network, this gives a solution time of 278 minutes; for a 15-pole, 538 minutes; for a 20-pole, 945 minutes.

4. The Network Synthesis Library

Figure 6 shows the block diagram of a network synthesis library for the automatic computation of the approximation functions for a variety of filters. This diagram is by no means considered to be all-inclusive, it is only intended to show how such a library would be assembled.

It should be obvious from the diagram how our various blocks are used together to provide a large number of filter characteristics from a relatively small number of programs. All blocks would have common input, output and carry* locations, and programs represented by the blocks would have bootstrapping links to call in the next block in order to proceed with the computation.

In addition to the "frequency" filters shown, we would consider it desirable to provide in the basic library:

a. Tschebycheff Stop-Band Ripple Filters

b. Tschebycheff Equi-Ripple Filters

With regard to the latter, Byron J. Bennett[1] has developed an interesting iterative technique which seems to be much simpler than Alexander J. Grosman's[2] method for avoiding the elliptic function.

Note the provision for the realizibility check. It is considered that the inclusion of such a routine is mandatory in order to save computer time. This routine would compute n,[3] the number of elements required to

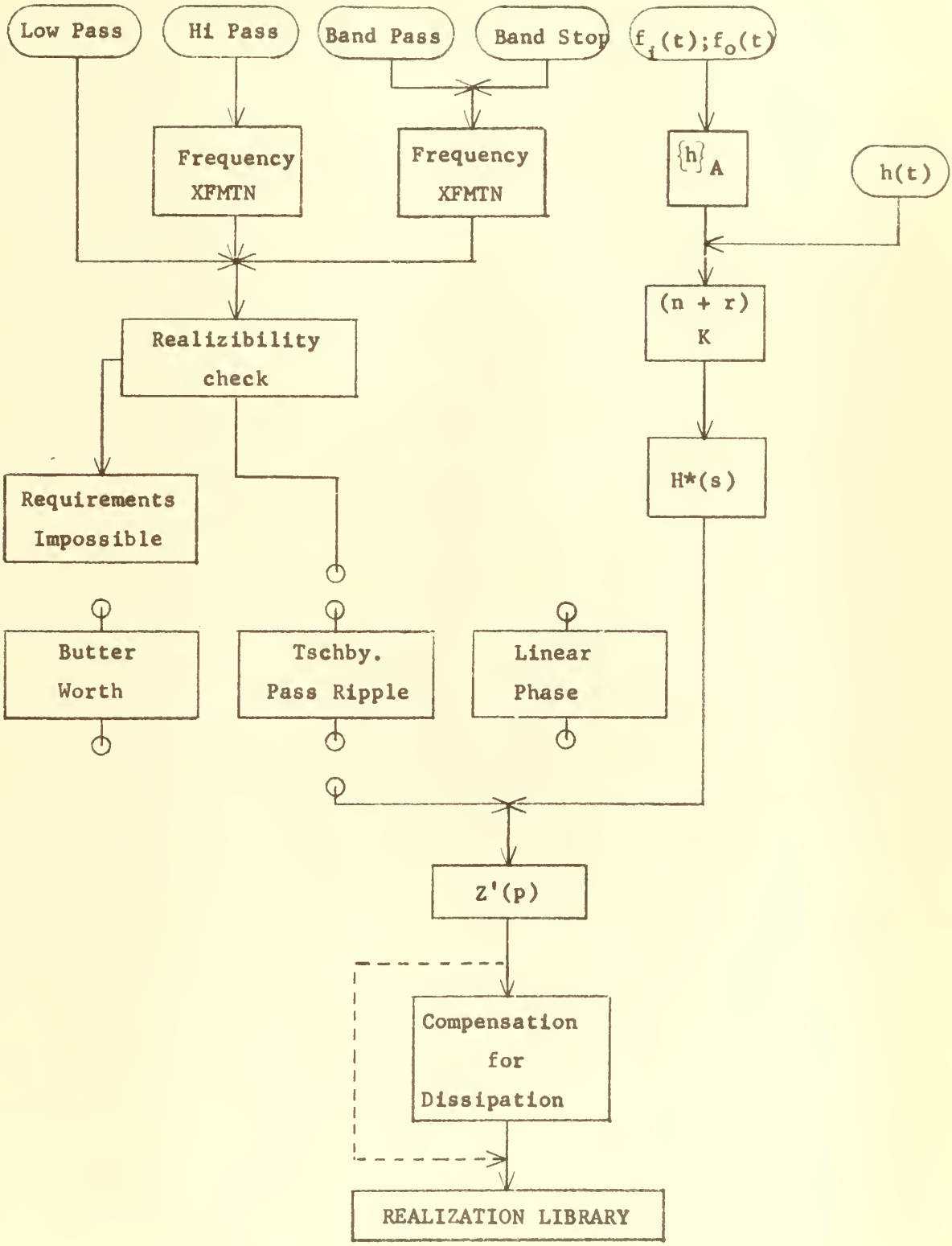
*We use the term carry to characterize that data which is not used in the present computation but must be "carried" for use in later routines.

[1]Byron J. Bennett, A Note on Circuit Synthesis, p. 61.(16)

[2]A. J. Grosman, Synthesis of Tschebycheff Parameter Symmetrical Filters, Proc. IRE, Vol. 45, No. 4, pp. 454-474.(13)

[3]Using Grosman's(13) equation (38) or see Darlington(4).

provide the required selectivity parameter (or discrimination parameters) and meet the ripple requirements, if n is specified and the requirements demand more elements than is allowed the computer should so indicate to the operator. It is estimated that this information would be available within not more than ten minutes from the start of computations, and therefore establish a doctrine that if such a program is used, the operator must wait until this point in the program is passed before allowing the computer to proceed unattended, and by so doing be assured that the results will be realizable as a physical network, and that computation time will not be wasted.



BASIC NETWORK SYNTHESIS LIBRARY

Figure 6

5. Conclusions

The results of the tests on the routines programmed to date indicate that a library of network synthesis routines employing conventional techniques is practicable.

The use of such a library should result in considerable savings in time and effort for those people concerned with network design.

The accuracy obtainable is more than sufficient to guarantee excellent approximations (and undoubtedly synthesis) if the number of elements to be used is not severely restricted.

The time required for solution is not excessive in comparison with the magnitude of the problem and the accuracy of the results. The employment of the library of programs will require a working knowledge of the computer, unless magnetic tapes are made up from the "building blocks" to provide specific types of network designs. In this case, detailed instructions may suffice to permit persons not familiar with the computer to use the programs to good advantage.

The installation of the CDC - MK 1 Computer will enhance the value of this library despite the conversion problem. The increased speed of computation and shorter programs which will be possible due to the "built-in" floating point arithmetic will make such a library even more valuable.

BIBLIOGRAPHY

1. E. A. Guillemin, Communications Networks, Vol. II, John Wiley and Sons, Inc., New York, 1935.

2. E. A. Guillemin, Synthesis of Passive Networks, John Wiley and Sons, Inc., New York, 1957.

3. Freddy Ba Hli, Network Synthesis by Impulse Response for Specified Input and Output in the Time Domain, Technical Report No. 261, M.I.T. Research Laboratory of Electronics, July 31, 1953.

4. S. Darlington, Synthesis of Reactance Four-Poles, Journal of Mathematics and Physics, Vol. XVIII, 1939.

5. E. A. Guillemin, A Summary of Modern Methods of Network Synthesis, Advances in Electronics, Vol. III, Academic Press, Inc., New York, 1951.

6. M. F. Gardner, J. L. Barnes, Transients in Linear Systems, John Wiley and Sons, Inc., New York, 1942.

7. Proceeding of Symposium on Modern Network Synthesis, Vol. V, Polytechnic Institute of Brooklyn, 1956.

8. D. R. Hartree, Numerical Analysis, Oxford, 1952.

9. A. S. Householder, Principles of Numerical Analysis, McGraw-Hill, New York, New York, 1953.

10. J. B. Scarborough, Numerical Mathematical Analysis, John Hopkins Press, London, 1950.

11. D. P. MacCracken, Digital Computer Programming, John Wiley and Sons, Inc., New York, 1957.

12. K. S. Kunz, Numerical Analysis, McGraw-Hill Book Company, Inc., New York, 1957.

13. A. J. Grosman, Synthesis of Tschebycheff Parameter Symmetrical Filters, Proceedings of the I.R.E., Vol. 45, No. 4, pp. 454-474.

14. T. R. Bashkow and C. A. Desoer, Digital Computers and Network Theory, Bell Telephone Laboratories, Inc., Wes Con Record, August, 1957.

15. Wataru Mayeda, Digital Determination of Topological Quantities and Network Functions, Electrical Engineering Research Laboratory, University of Illinois, Urbana, Illinois, 1957.

16. Transactions of the I.R.E., Professional Group on Circuit Theory, Vol. CT-1, No. 3, September, 1954.

17. E. A. Guillemin, The Mathematics of Circuit Analysis, John Wiley and Sons, Inc., New York, New York, 1949.

18. D. E. Muller, Solving Algebraic Equations by an Automatic Computer, Mathematical Tables and Other Aids to Computation, Vol. X, No. 56, October, 1956, pp. 208-215.

19. W. E. Milne, Numerical Calculus, Princeton University Press, Princeton, New Jersey, 1949.

20. W. K. Linvill, Use of Sampled Functions for Time Domain Synthesis, Proceedings of N.E.C., Vol. IX, February 1954, pp. 533-542.

21. Nick de Claris, An Approximation Method with Rational Functions, Technical Report No. 287, M.I.T. Research Laboratory of Electronics, December, 1954.

22. George B. Dantzig, Computational Algorithm of the Revised Simplex Method, USAF Project RAND Memorandum, ASTIA Document No. AD 114135, 26 October 1953.

23. H. M. Markowitz, The Elimination Form of the Inverse and Its Application to Linear Programming, Management Science, Vol. 3, No. 3, April, 1957.

24. John von Neumann and H. H. Goldstine, Numerical Inverting of Matrices of High Order, American Mathematical Society Bulletin, Vol. 53, p. 1096.

25. Z. Kopal, Numerical Analysis, John Wiley and Sons, Inc., New York, 1955.

APPENDIX I

It is the intention of this appendix to explain with a minimum of theoretical reasoning the overall flow chart of Ba Hli's method.

The following inputs are required:

1. $\{f_o(t)\}$ - time sequence of output function (in octal or decimal).
2. $\{f_i(t)\}$ - time sequence of input function (in octal or decimal).
3. N_d - number of discontinuities.
4. N_s - number of points of inflection.
5. N_e - number of elements.
6. t - number of $f_i(t)$ ordinates used.
7. Octal or Decimal input data indicator.

Annex 1 shows a block diagram flow chart which considers only inputs and outputs to major program subdivisions, and major logical decisions required.

We allow for either decimal or octal digit input sequences, in either case they are converted to binary floating point. If the number of elements (N_e) is not specified, compute AS = 1.5 N_d + N_s (see page 17,main text).

The synthetic division process is self-explanatory, the quantity "t", the number of $f_i(t)$ ordinates, is required in this block.

Annex 2 shows a detailed flow chart for the determination of "r", the relative degree between numerator and denominator. The first order approximation used to compute h_0 is considered to be sufficiently accurate to ascertain whether or not h(t) starts from a finite value, or from zero. The "0" used in the decision boxes is actually slightly greater than zero, about .025, to avoid any erroneous decision that h_0 is greater

than zero, when it is only greater due to the error introduced by the first order approximation. Note that we have deviated from Ba Hli's method of determining the behavior of h(t) near t = 0 in that we compute derivatives.

We can now determine, D_d, the degree of the denominator, and N_d, the degree of the numerator of Z(p) and the number of terms of H*(s), H_d, we must compute to solve for Z(p).

We can make a somewhat arbitrary decision as to how accurately we should compute our H(s), i.e., what subdivisions we should use in our "moment" computing scheme (see Appendix II), depending on the number of terms we shall have in our denominator.

The next two appendices give the program details for the solution of H*(s), and Z'(p).

The solution of Z(p) is now complete, unless we desire to compensate for dissipative losses. This is easily accomplished after factoring the numerator and denominator polynominals. We preshift our pole and zero pattern to the right an amount "α", unless this shift causes a pole or zero to move into the right half plane and thereby violate the conditions of physical realizability. We compute

$$\alpha = R/2L$$

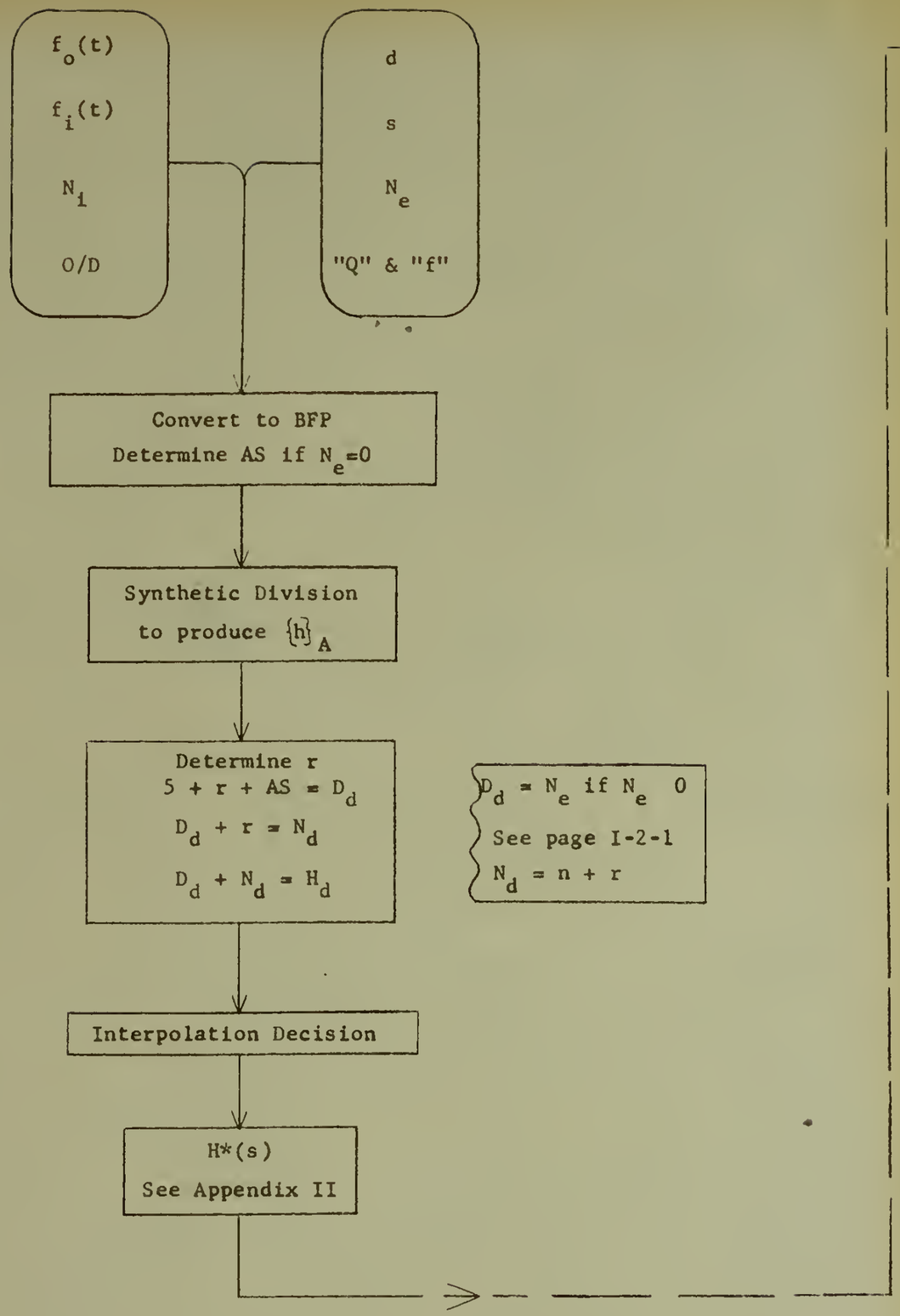
$$2L/R = 1/\alpha$$

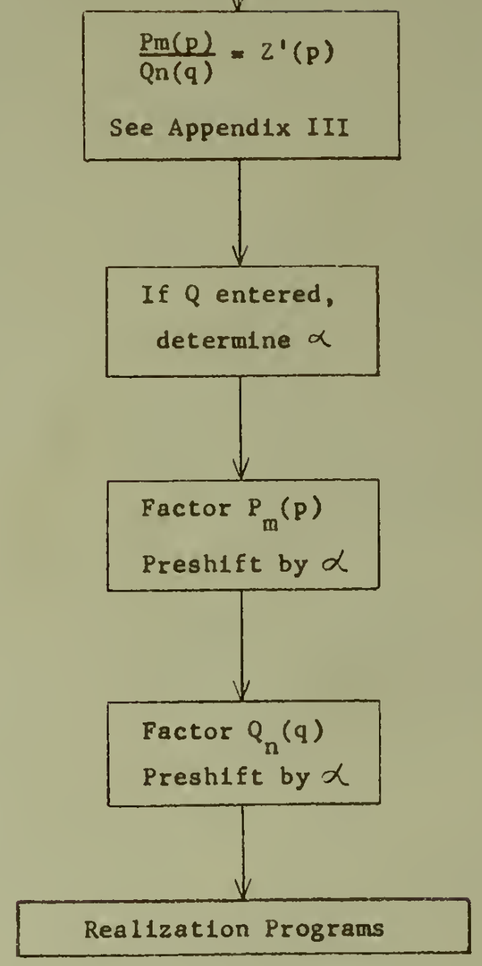
$$\omega L/R = \omega/2\alpha$$

$$2Q/\omega = \alpha$$

$$\alpha = Q/\pi f$$

and thus we compute α as a function of Q and f, where f is the lowest significant frequency involved and is entered by the designer as input data.



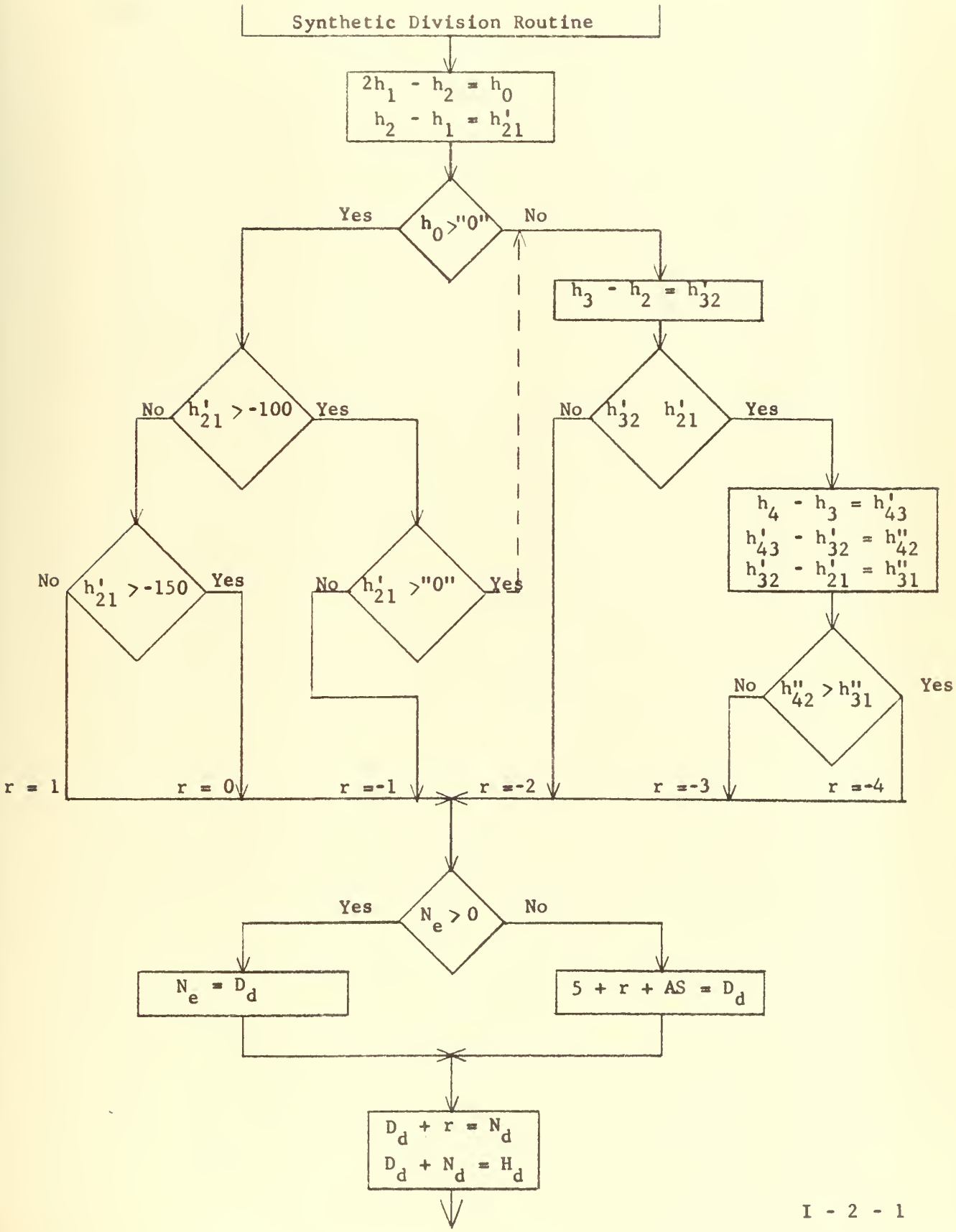


OVERALL FLOW CHART

OF

TIME DOMAIN NETWORK SYNTHESIS PROGRAM

Selection of Relative Degree



APPENDIX II

CALCULATION OF H*(s) FROM THE $\{h\}_A$ SEQUENCE

We are now required to form the power series expansion of:

$$\sum_{i=1}^{s} a_i e^{-T_i}$$

As was mentioned in the main body of this paper, replacing the Laplace-Stieljes integral by the summation indicated above, resulted in an increasing error in those terms higher than the first degree. We recognize that this error results from the failure to compute the exact moment of $\{h\}_A$ about the origin.

The true j^{th} moment of an area defined by f(x) is:

$$M_j = \int_0^{\infty} x^j f(x)\, dx$$

which we have replaced by the summation

$$M_j(app) = \sum_{i=1}^{s} \left[\frac{2_i - 1}{2N}\right]^j A_i$$

where A_i = incremental area or h_{a_i} and S = NK (K = 1,2,3....)

For our program, we limit K to 4; that is, we require the impulse response to be defined over the range 0 to 4 time units. The input and output functions may be sealed so this will be adequate for most problems.

It might be well before proceeding to take a close look at the manner in which we form the coefficients for the Dirichlet power series expansion.

We may proceed in such a manner as to form one term at a time, that is, referring to Figure II-1, we may sum first down the j = 0 column, then the j = 1 column, etc. The operation count for this procedure is:

N "a" for the first term,

N "a" + N"m" + "m" for the second (if we factor out the 1/2N until the sum is complete),

N "a" + N(j + 1)"m" + (j + 1)"m" for the j^{th} term.

a total of

$$(j + 1)\ N"a" + N+1 \sum_{i=1}^{j+1} (i + 1)\ "m"$$

$$(j + 1) \left\{ N \left["a" + \left(\frac{j+2}{2} + 1 \right) "m" \right] + \left(\frac{j+2}{2} + 1 \right) "m" \right\}$$

where "a" stands for floating point addition and "m" stands for floating point multiplication.

For N = 40, j = 20; this becomes 840 "a" + 11040 "," and if we assume that floating point addition requires twice the time that floating point multiplication does, a equivalent total of 12,720 floating point multiplications.

If, however, one proceeds out the rows and forms all the partial sums associated with A_1, then A_2, etc., for i = 1, j = 0, to j, the count per row is:

"a", 2"m" + "a", "a" + "m", "a" + "m", ... "a" + "m" and the 2N in the denominator is included.

The count is the same for all rows, therefore, the total is:

N(j+1)("a"+"m"), which for N = 40, j = 20, is 840 "a" + 840 "m", or based on our previous assumption, a total of

2,520 floating point operations.

There is, however, some additional "call-up" and "put-away" address computation which degrades this efficiency somewhat. Nonetheless, it seems quite evident that a saving of at least 50% can be effected in this way.

With this saving we can afford to reduce the error in the final coefficients by introducing an interpolation routine which effectively

doubles or even quadruples the number of sample points used.

The interpolation precedure makes a straight line approximation between sample points, and computes the value the intermediate sample points would have. Therefore, it is able to use a smaller increment on the abscissa and thus approaches a true integration more closely than would be otherwise possible. The results of some runs using various "degrees" of interpolation are presented in Table II-1 for comparison purposes. The flow chart and program are appended hereto.

i \ j	0	1	2	3	.	j
1	A_1	$\frac{1}{2N} A_1$	$(\frac{1}{2N})^2 A_1$	$(\frac{1}{2N})^3 A_1$	.	$(\frac{1}{2N})^j A_1$
2	A_2	$\frac{3}{2N} A_2$	$(\frac{3}{2N})^2 A_2$	.	.	.
3	A_3	$\frac{5}{2N} A_3$	$(\frac{5}{2N})^2 A_3$	.	.	.
4	A_4	$\frac{7}{2N} A_4$	.	.	.	.
5	A_5	$\frac{9}{2N} A_5$	.	.	.	.
.	.	.	.	.	.	.
.	A_i	$(\frac{2i-1}{2N})A_i$	$(\frac{2i-1}{2N})^2 A_i$	$(\frac{2i-1}{2N})^3 A_i$	.	$(\frac{2i-1}{2N})^j A_i$
.	.	.	.	.	.	.
N	A_n	$(\frac{2N-1}{2N})A_n$	.	.	.	.

$$H^*(s) = \sum_i A_i + \sum_i (\frac{2N-i}{2N}) A_i + \sum_i (\frac{2N-i}{2N})^2 A_i + \sum_i (\frac{2N-1}{2N})^3 A_i + \ldots + \sum_i (\frac{2N-1}{2N})^j A_i$$

The factorials which appear in H(s) are yet to be supplied, and are brought in with the alternating sign according to whether j is odd or even.

Formation of H*(s)

Figure II-1

TABLE II-1(A)

j	Actual	N = 40	E	N = 80	E	N = 160	E
0	1.000000000	0.999999999	nil	0.999999999	nil	0.999999999	nil
1	0.500000000	0.499999999	nil	0.500078124	1.56	0.500098755	2.98
2	0.166666666	0.166640624	1.56	0.166661132	.33	0.166666259	.24
3	0.041666666	0.041653645	3.13	0.041663418	.78	0.041665861	.19
4	0.008333333	0.008328993	5.21	0.008332248	1.30	0.008333062	.33
5	0.001388888	0.001387804	7.78	0.001388617	1.95	0.001388821	.48
6	0.0001984127	0.000198195	10.99	0.000198358	2.77	0.000198399	.71
7	0.00002480159	0.0000024765	14.75	0.0000248068	3.52	0.000024799	1.04
8	0.0000027558	0.0000027500	21.46	0.0000027540	6.51	0.000002755	2.90
9	0.0000002756	0.0000002750	---	0.0000002750	---	0.000000275	---

Comparison of Results between the True Laplace Expansion and Ba Hli's Approximation Method for determining $H^*(s)$ using 40, 80, and 160 subdivisions, for $H(t) = 1 \quad 0 \leq t \leq 1$

0 elsewhere

$$E = \left[\frac{\text{true value - computed value}}{\text{true value}}\right] \times 10^4$$

TABLE II-1(B)

j	Actual	N = 40	E	N = 80	E	N = 160	E
0	.2500000000	.250000000	nil	.249999999	nil	.249999999	nil
1	.1250000000	.124999999	nil	.125000976	.078	.125001220	.0976
2	.03645833333	.036464843	1.786	.036460461	.584	.036459365	.280
3	.00781250000	.007815755	4.166	.007813442	1.210	.007812863	.465
4	.00134548600	.001346435	7.053	.001345745	1.925	.001345572	.637
5	.0001953125	.000195515	10.407	.000195366	2.713	.000195328	.769
6	.000024607825	.000024642	14.219	.000024616	3.252	.000024610	.813
7	.000002744967	.000002750	18.454	.000002746	4.007	.00002745	...
8	.000000275035	.000000275	...	...	...	...	
9	.0000000250275	.000000250	...	...	...	...	
	Time Required	11 mins		21 mins		42 mins	

Comparison of Results between the True Laplace Expansion and Ba Hli's Approximation for determining H*(s), using 40, 80, and 160 subdivisions for $H(t) = t \quad 0 \leq t \leq 1/2$

$= 1-t \quad 1/2 \leq t \leq 1$

$= 0 \quad$ elsewhere

$$E = \left[\frac{\text{true value - computed value}}{\text{true value}}\right] \times 10^4$$

DIRICHLET POWER SERIES EXPANSION FOR PSEUDO-SYSTEM FUNCTION

ADD	INST	M^1	M^2	M^3	REMARKS
0300	05	3000	3000	0550	Load buffer
0301	34	3000	2100	2000	and clear putaways
0302	17	2010	3000	0305	If TS 1 up, 0305 (1)
0303	30	0556	2100	0545	No, Set t to 1
0304	34	3000	2100	0311	and skip to 0311
0305	17	2020	3000	0310	If TS 2 up, 0310
0306	30	0541	2100	0545	No, Set t to 2
0307	34	3000	2100	0311	and skip to 0311
0310	30	0542	2100	0545	Set t to 3
0311	35	0536	0545	0535	N exp + t to N' exp
0312	32	0510	0516	0314	Set h_1 pickup
0313	31	0533	0574	0531	"-1" to SA
0314	31	0120	0121	0523	h_1 to TF
0315	34	0545	0532	0321	If t > 1, 0321
0316	31	0523	0564	0530	TF to TA
0317	30	0533	2100	0547	1 to H
0320	34	3000	2100	0413	Skip to 0413
0321	34	0545	0534	0350	If t > 2, 0350
0322	33	0572	0533	0326	No, if SA > "0", 0326
0323	36	0523	0533	0521	No, TF/2 to Δ
0324	30	0564	2100	0562	
0325	34	3000	2100	0333	Skip to 0333
0326	31	0523	0564	2000	Load TF
0327	31	0524	0565	2002	Load TE
0330	34	3000	2100	1516	TF - TE
0331	36	2000	0534	2000	(TF - TE)/4
0332	31	2000	2001	0521	(TF - TE)/4 to Δ
0333	31	0523	0564	2000	Load TF
0334	31	0521	0562	2002	Load Δ
0335	34	3000	2100	1516	TF - Δ
0336	36	2000	0532	2000	(TF - Δ)/2
0337	31	2000	2001	0530	to TA
0340	31	0523	0564	2000	Load TF
0341	31	0521	0562	2002	Load Δ
0342	34	3000	2100	1540	TF + Δ
0343	36	2000	0532	2000	(TF + Δ)/2
0344	31	2000	2001	0527	to TB
0345	31	0523	0564	0524	TF to TE
0346	30	0534	2100	0547	2 to H
0347	34	3000	2100	0413	Skip to 0413
0350	33	0572	0533	0356	Is "SA" > "0"
0351	36	0523	0534	0521	No, TF/4
0352	30	0564	2100	0562	to Δ
0353	36	0523	0532	0522	TF/2
0354	30	0564	2100	0563	to α
0355	34	3000	2100	0365	Skip to 0365
0356	31	0523	0564	2000	Load TF
0357	31	0524	0565	2002	Load TE

(1) The test switch function would be performed by the interpolation selection in the overall synthesis program.

DIRICHLET POWER SERIES EXPANSION FOR PSEUDO-SYSTEM FUNCTION

ADD	INST	M^1	M^2	M^3	REMARKS
0360	34	3000	2100	1516	TF - TE
0361	36	2000	0534	2000	(TF - TE)/4
0362	31	2000	2001	0522	to α
0363	36	2000	0532	2000	(TF - TE)/8
0364	31	2000	2001	0521	to Δ
0365	31	0523	0564	2000	Load TF
0366	31	0522	0563	2002	Load α
0367	34	3000	2100	1516	TF - α
0370	31	0521	0562	2002	Load Δ
0371	34	3000	2100	1516	TF - α - Δ
0372	36	2000	0534	0530	(TF - α - Δ)/4
0373	30	2001	2100	0571	to TA
0374	31	0522	0563	2002	Load Δ
0375	34	3000	2100	1540	TF- Δ
0376	36	2000	0534	0527	(TF - Δ)/4
0377	30	2001	2100	0570	to TB
0400	31	0523	0564	2000	Load TF
0401	31	0521	0562	2002	Load Δ
0402	34	3000	2100	1540	TF + Δ
0403	36	2000	0534	0526	(TF + Δ)/4
0404	30	2001	2100	0567	to TC
0405	31	0522	0563	2002	Load α
0406	34	3000	2100	1540	TF + Δ + α
0407	36	2000	0534	0525	(TF + Δ + α)/4
0410	30	2001	2100	0566	to TD
0411	31	0523	0564	0524	TF to TE
0412	30	0543	2100	0547	Set H to 4
0413	31	0531	0572	2000	Load SA
0414	31	0534	0575	2002	Load "2"
0415	34	3000	2100	1540	SA + 2
0416	31	2000	2001	0531	to SA
0417	31	0535	0576	2004	Load N'
0420	34	3000	2100	1507	Divide
0421	31	2000	2001	0520	Result to M
0422	32	0460	0516	0433	Reset putaway
0423	33	0520	2100	0457	If M $\geq$ 0, 0457
0424	35	2100	2100	0544	No, clear j tally
0425	34	0544	2100	0427	If j $\geq$ 0, skip to 0427
0426	34	3000	2100	0433	No, skip to 0433
0427	31	0530	0571	2000	Load TA
0430	31	0520	0561	2004	Load M
0431	34	3000	2100	1500	TA X M
0432	31	2000	2001	0530	to TA
0433	31	0600	0601	2000	Load summation
0434	31	0530	0571	2002	Load TA
0435	34	3000	2100	1540	Accumulate
0436	30	0433	1505	2006	Build putaway
0437	32	2006	1677	0442	Build putaway

DIRICHLET POWER SERIES EXPANSION FOR PSEUDO-SYSTEM FUNCTION

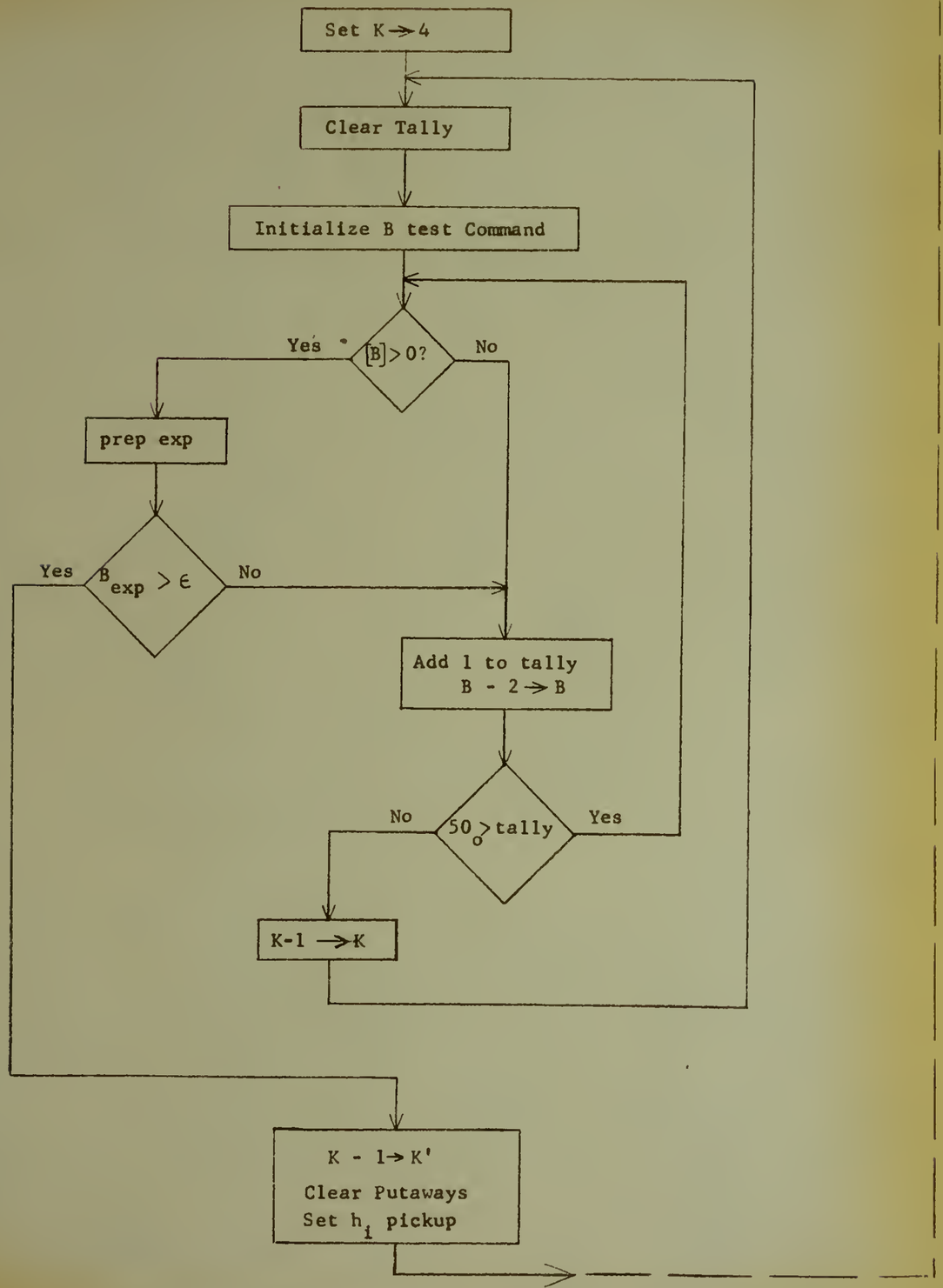
ADD	INST	M^1	M^2	M^3	REMARKS
0440	35	0442	0532	2006	Build putaway
0441	32	2006	1677	0443	Build putaway
0442	30	2000	2100	0616	Putaway
0443	30	2001	2100	0617	Putaway
0444	35	0544	0532	0544	Add 1 to j tally
0445	35	0433	0512	0433	Modify summation pickup
0446	34	0540	0544	0425	If J $\geq$ j, repeat to 0425
0447	36	0547	0532	0547	No, reduce H by 1
0450	34	0547	2100	0453	If H $\geq$ 0, skip to 0453
0451	35	0314	0512	0314	No, modify h_i pickup
0452	34	3000	2100	0314	Repeat loop
0453	31	0527	0570	0530	Shift TB to TA
0454	31	0526	0567	0527	TC to TB
0455	31	0525	0566	0526	TD to TC
0456	34	3000	2100	0413	and repeat loop
0457	32	0514	0516	0464	Reset h_i pickup
0460	31	0600	0601	1310	First term pickup
0461	34	3000	2100	1011	Convert and print H_o (2)
0462	30	0532	2100	0544	Set j to 1
0463	31	0532	0573	0517	Load F!
0464	31	0620	0621	2000	Pickup H_i term
0465	32	0544	0532	2106	Extract last bit of j
0466	34	2006	2100	0470	If j odd, skip to 0470
0467	34	3000	2100	0471	No, skip to 0471
0470	32	0441	0537	2001	Change sign to minus
0471	31	0517	0560	2004	Load F!
0472	34	3000	2100	1507	Divide by F!
0473	31	2000	2001	1310	Load H
0474	34	3000	2100	1011	for conversion and print out[1]
0475	35	0544	0532	0544	Increment j by 1
0476	31	0551	0544	2000	Load "j"
0477	31	0517	0560	2004	Load F!
0500	34	3000	2100	1500	Multiply for (F + 1)!
0501	31	2000	2001	0517	Putaway
0502	35	0464	0512	0464	Modify H_i pickup
0503	34	0540	0544	0464	Have J terms been divided by appropiate factorial; no, 0464
0504	22	0000	0000	0000	Yes, halt
0505 - 507		Available			
0510	00	0000	0001	0000	h_i pickup reset
0511	35	2000	2005	2000	Putaway clear routine
0512	00	0002	0002	0000	Modifier for 0502
0513	34	3000	2100	0302	Putaway clear routine
0514	00	0602	0603	0000	Putaway reset
0515	00	0000	0000	0001	Putaway clear routine
0516	00	7777	7777	0000	M^1 and M^2 extractor
0517 - 0536		Used for tempoary storage			
0537	02	0000	0000	0000	Sign extractor
0540	00	0000	0000	0000	J

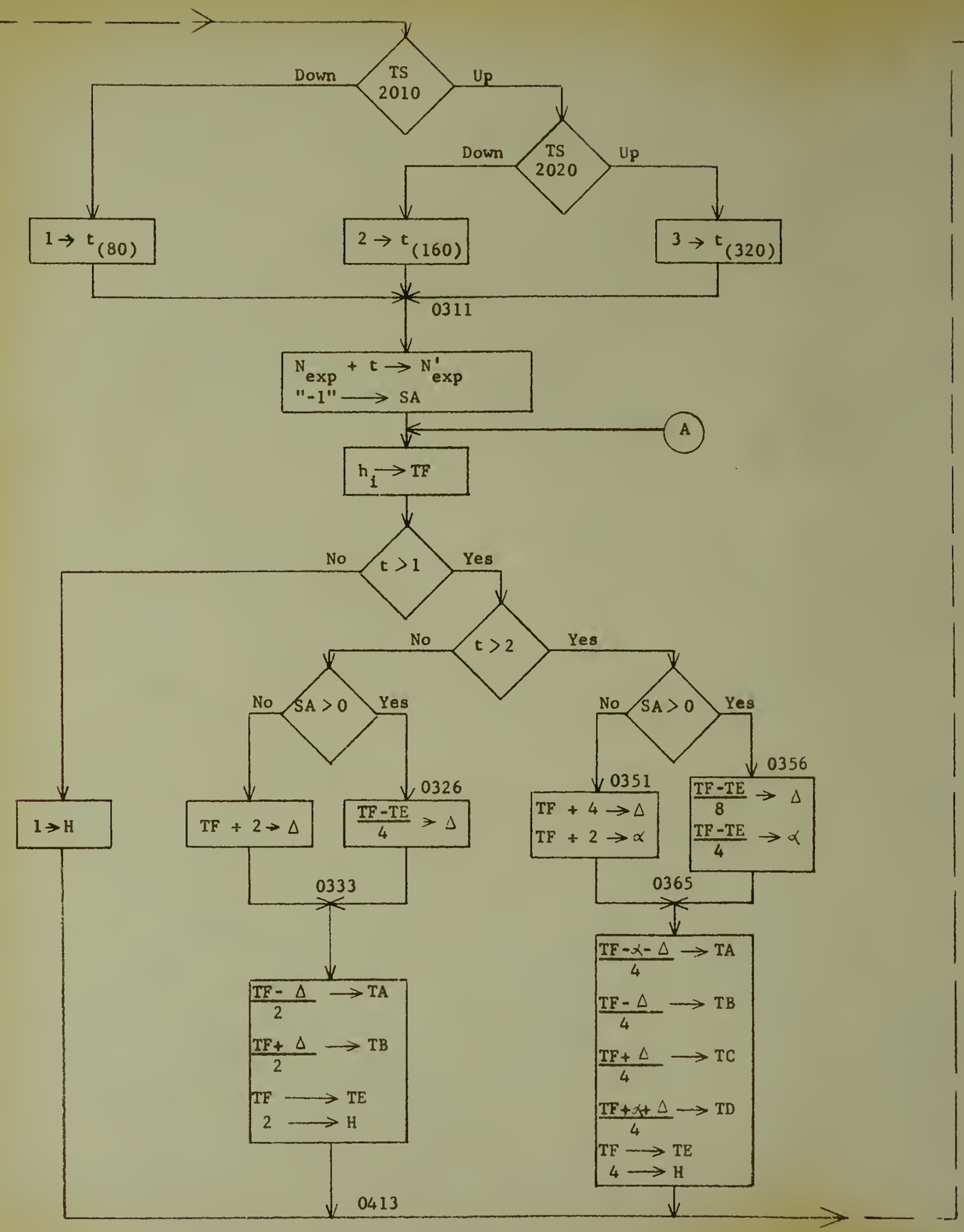
DIRICHLET POWER SERIES EXPANSION FOR PSEUDO-SYSTEM FUNCTION

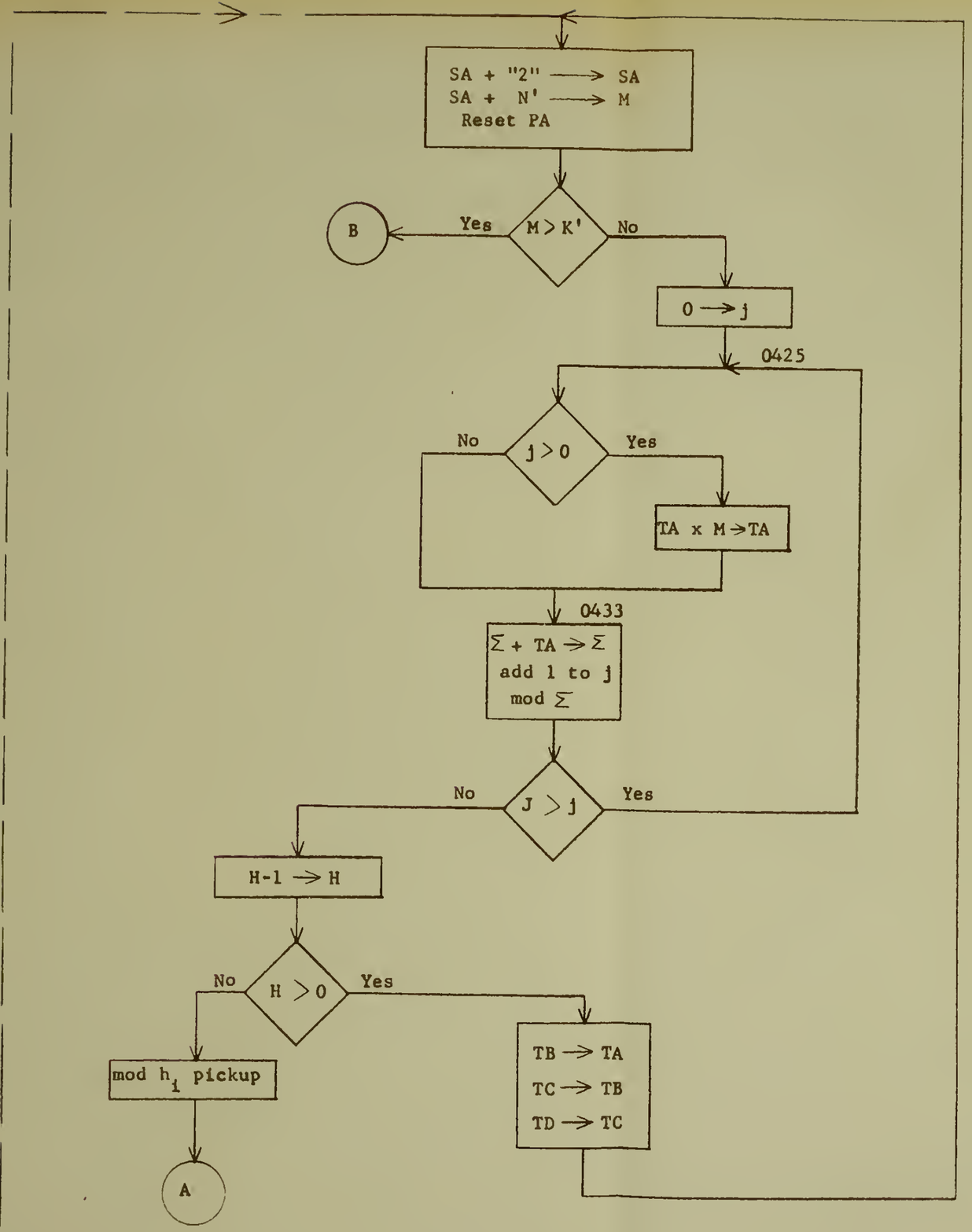
ADD	INST	M^1	M^2	M^3	REMARKS
0541	00	0000	0000	0002	2 - for H
0542	00	0000	0000	0003	3 - for H
0543	00	0000	0000	0004	4 - for H
0544	00	0000	0000	0000	j tally
0545	00	0000	0000	0000	t storage
0546	00	0000	0000	0001	1
0547	00	0000	0000	0000	H storage
0550	26	2100	2100	0600	Putaway clear routine
0551	00	0000	0000	0044	Constant for 0476
0552	34	2004	2000	2000	Putaway clear routine
0553	00	0000	0000	0000	Available
0554	00	2100	2100	1000	Testword for clear
0555 - 0577		Used for temporary storage			
0600 - 1000		Used for storage of results (may be reduced to provide a minimum of 2(J + 1) cells.			

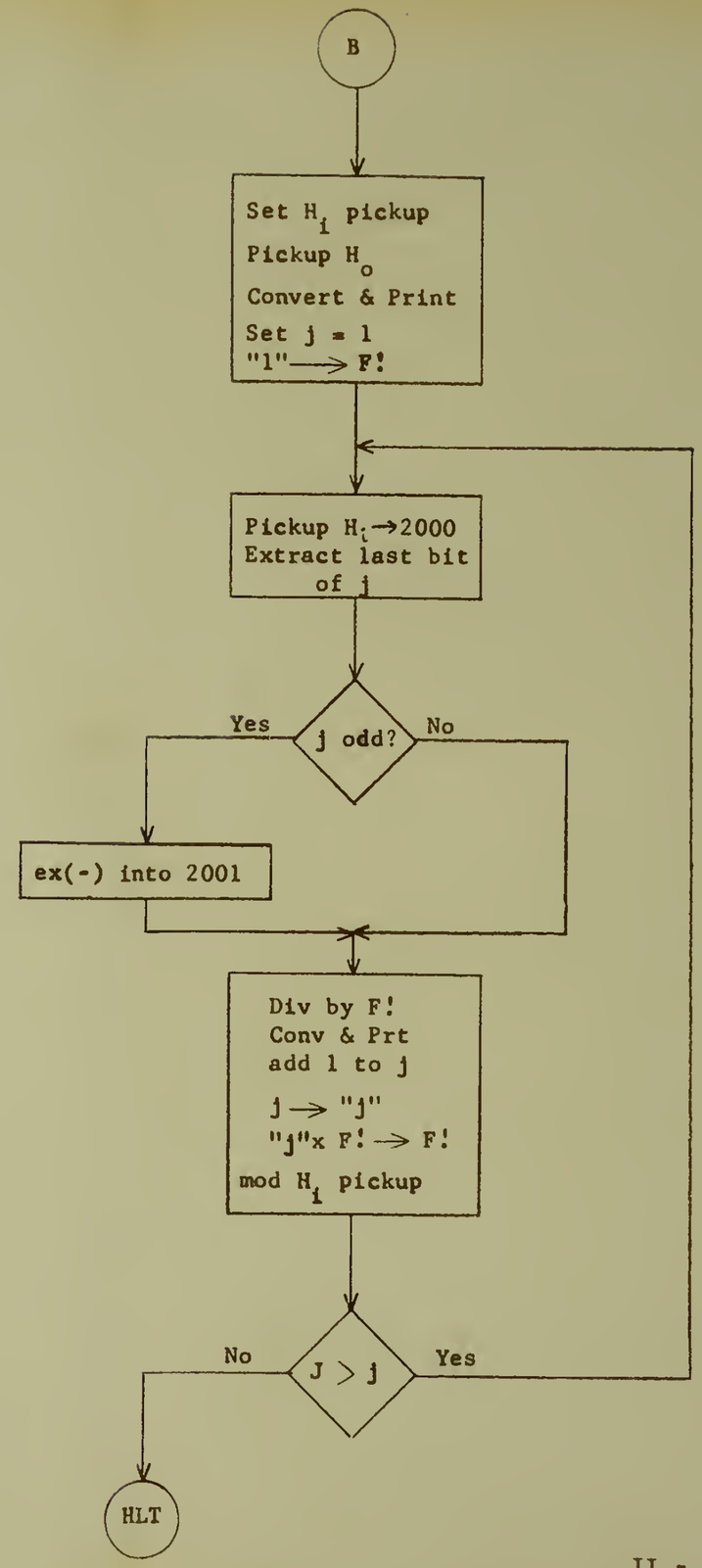
(2) The conversion and print routine is linked in this program for test purposes only and should be replaced with a boot-strapping routine to call in next portion of the overall program.

FLOW CHART OF DIRICHLET POWER SERIES EXPANSION






SA + "2" → SA
SA + N' → M
Reset PA
B
Yes
M > K'
No
0 → j
0425
No
j > 0
Yes
TA x M → TA
0433
Σ + TA → Σ
add 1 to j
mod Σ
No
J > j
Yes
H-1 → H
No
H > 0
Yes
mod h_i pickup
A
TB → TA
TC → TB
TD → TC


B
Set H_i pickup
Pickup H_o
Convert & Print
Set j = 1
"1" → F!
Pickup H_i → 2000
Extract last bit
of j
Yes
j odd?
No
ex(-) into 2001
Div by F!
Conv & Prt
add 1 to j
j → "j"
"j"x F! → F!
mod H_i pickup
No
J > j
Yes
HLT

APPENDIX III

MATRIX INVERSION

As was noted on pages 14 and 15 of the main text of this paper, the final solution of Ba Hli's method led to a system of simultaneous linear equations. While it is certainly possible to program the solution of a system of equations by the straight forward elimination technique, this method is awkward when many equations are involved (and for precise synthesis of transcendental functions, there must be many). The computer must do all the address computation based on N, the number of poles (or the degree of the denominator) and r, the relative degree between numerator and denominator. This method for n equations in n unknowns, neglecting the address computations, would require n^2 divisions to eliminate the first unknown, $(n-1)^2$ to eliminate the second, etc. In all, to reduce the system to the desired "one equation in one unknown" (we are dealing with square matrices), requires:

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \text{ operations*}$$

If, and only if, we have saved the intermediate equations, we may complete the solution in

$$\frac{n^2+n}{2} \text{ operations}$$

or for the complete solution,

$$\frac{n^3}{3} + n^2 + \frac{2n}{3} \text{ operations}$$

Such a procedure would require an inordinate amount of storage

*The term "operation" is customarily taken to mean multiplications or divisions. Addition, subtractions and address modifications are not included in operation counts.

space and worse, intricate address computations.

For simplicity of programming and best relative speed, we would suggest that the denominator matrix, i.e., that group of equations which are equated to zero, and from which we determine the q's, be solved by the elimination form of the inverse,[1,2] and solve the remaining equations for p_0, p_1.....p_m, which incidentially will always be explicit in terms of the q's, by simple algebra.

We will describe this highly efficient method of computing an inverse matrix and demonstrate its simplicity and economy of operation by means of a simple example. Briefly, by way of introduction, we have the situation:

$[H]\ [Q] = [h]$, if we multiply by the inverse $[H]^{-1}$

$[H]^{-1}\ [H]\ [Q] = [H]^{-1}\ [h]$ we have

$[Q] = [H]^{-1}\ [h]$, the desired result.

Our immediate problem is to find a rapid means of computing the inverse of H. The general method of solution can be summarized:

(1) $[H\ \ I]$ This operation can be carried out by our

(2) $[H^{-1}\ H\ \ H^{-1}\ I]$ suggested technique operating solely on

(3) $[I\ H^{-1}]$ the unit matrix.

We perform this transformation on I by bringing in H one column at at time, multiplying by those elements which have been transformed by the introduction of the columns preceding.

[1]George B. Dantzig, Computational Algorithm of the Revised Simplex Method, USAF Project Rand Memorandum, ASTIA Document No. AD 114135, 26 Oct., 1953.

[2]H. M. Markowitz, The Elimination Form of the Inverse and Its Application to Linear Programming, Management Science, Vol. 3, No. 3, April, 1957.

The actual transformation may be summarized

(1) H_{jk} $\quad j,k = 1,2....n$

(2) I_{jk}

(3) $I_{jk}\ H_k = P_k$

(4) $I'_{jk} = I_{jk} \div P_{kr} \quad (j = r) \quad k = 1,2....n$

(5) $I'_{jk} = I_{jk} - P_{kj}\ I'_{rk} \quad (j \neq r)$

which states (for those not versed in matrix notation), given a n x n matrix H (1), construct the corresponding n x n unit matrix I (2). Multiply the $k^{\underline{th}}$ column of H by the unit matrix (or the partly transformed matrix) to produce the pivotal column P_{kj}, (3). Divide the elements of the $r^{\underline{th}}$ row of the unit or matrix under transformation by $r^{\underline{th}}$ element of the pivotal column, (4). Correct all other rows by subtracting from the element to be corrected the product of pivotal element in that row and the corresponding element in the $r^{\underline{th}}$ row computed in Step (4), (5).

A simple example will make the procedure clear, say H is

$$\begin{bmatrix} 4 & 2 & 3 \\ 1 & 0 & 2 \\ 2 & 1 & 2 \end{bmatrix}$$

We start by writing down the unit matrix of corresponding order, viz.,

(2) $$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$K_r = 1$, $R = 1$

(3) $$\begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 \end{bmatrix}$$

Note: This is akin to merely augmenting the unit matrix by H_1 for $R = 1$.

$K_r = 1$, $R = 1$

(4) $\begin{bmatrix} 1/4 & 0 & 0 & 4 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 \end{bmatrix}$ $\quad I'_{1k} = I_{1k} \div P_{1_1}$. $\quad k = 1,2....n$

$j = r = 1$

(5) $\begin{bmatrix} 1/4 & 0 & 0 & . \\ -1/4 & 1 & 0 & . \\ -1/2 & 0 & 1 & . \end{bmatrix}$ $\quad I'_{j1} = I_{j1} - P_{1j}\, I'_{1k}$ $(j \neq r = 1)$ $(k = 1)$

Discard P_{1_j}. Note: Columns to the right of I_{jk_r} are not operated on, herein is the great economy of this method.

$K_r = 2$, $R = 2$

(3) $\begin{bmatrix} 1/4 & 0 & 0 & 1/2 \\ -1/4 & 1 & 0 & -1/2 \\ -1/2 & 0 & 1 & 0 \end{bmatrix}$ $\quad I_{jk} \cdot H_2 = P_2$

(4) $\begin{bmatrix} 1/4 & 0 & 0 & 1/2 \\ 1/2 & -2 & 0 & -1/2 \\ -1/2 & 0 & 1 & 0 \end{bmatrix}$ $\quad J'_{2k} = J_{2k} \div P_{2_2}$ (second element in column P)

(5) $\begin{bmatrix} 0 & 1 & 0 & . \\ 1/2 & -2 & 0 & . \\ -1/2 & 0 & 1 & . \end{bmatrix}$ $\quad I'_{jk} = I_{jk} - P_{2_j}\, I'_{2k}$

Note: Again since $K = 2 < 3$, column 3 was unaffected; also note since $P_{2_3} = 0$, I_{3k} was unaffected.

$K = 3$, our final transformation is:

(3) $\begin{bmatrix} 0 & 1 & 0 & 2 \\ -1/2 & -2 & 0 & -5/2 \\ -1/2 & 0 & 1 & 1/2 \end{bmatrix}$ $\quad$ (4) $\begin{bmatrix} 0 & 1 & 0 & 2 \\ 1/2 & -2 & 0 & -5/2 \\ -1 & 0 & 2 & 1/2 \end{bmatrix}$ $\quad$ (5) $\begin{bmatrix} 2 & 1 & -4 \\ -4/2 & -2 & 5 \\ -1 & 0 & 2 \end{bmatrix}$

and therefore

$$H^{-1} = \begin{bmatrix} 2 & 1 & -4 \\ -2 & -2 & 5 \\ -1 & 0 & 2 \end{bmatrix}$$

Check:

$$[H]\,[H]^{-1} = [I]$$

$$\begin{bmatrix} 4 & 2 & 3 \\ 1 & 0 & 2 \\ 2 & 1 & 2 \end{bmatrix} \qquad \begin{bmatrix} 2 & 1 & -4 \\ -2 & -2 & 5 \\ -1 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The total number of operations (neglecting all incidental nulls) is:

Multiplications to bring in each new row	$n(n+1)(n+2)/3$
Multiplications and divisions to accomplish each transformation	$(n+1)(n)(n-1)/6$
To solve the equations once the inverse has been computed	n^2
Total product operations to complete solution	$\frac{n^3 + 4n^2 + n}{2}$

Reference to Table III-1 shows the efficiency of the elimination form of the inverse. It is important to note that by minimizing the number of operations, we also minimize the propagation of round-off error.

Von Neumann[1] has determined that an approximate inverse will be found if

$n < .15\,\beta^{s/4}$, where β is the base of the number system used in the computation, and S is the number of places. The program for the elimination form of the inverse (appended) used a packed floating point where the word structure was:

XS MMMM MMMM MQEE

Legend:
- X = not used
- S = sign of magnitude
- M = magnitude (octal digit)
- Q = 2 binary bits of magnitude & sign of exponent
- E = magnitude of exponent

[1] J. Von Neumann and H. H. Goldstine, Numerical Inverting of Matrices of High Order, American Mathematical Society Bulletin, Vol. 53, pp. 1096.

This realized an S of 29, and therefore theoretically the program should successfully compute the inverse for all matrices of $n < 23$.

The maximum round-off error to be expected is (after Von Neumann):

$$\epsilon = .5\beta^{-(S-1)}\left(\frac{n^3}{2} + 2n^2\right)$$

which for $\beta = 2$, $S = 29$, and $n = 25$ reduces to

$$\epsilon = \frac{18125}{2^{28}} = .000067$$

Thus we can expect an accuracy of, at worst, four significant figures for a 25 X 25 matrix.

If we decide to use a double precision floating point, S (for the CRC 102A) would increase to 66, by using the following configuration:

XS_1 MMMM MMMM MMMM XS_2 MMMM MMMM MMEE

where S_1 is sign of magnitude

S_2 is sign of exponent

and the round-off error now becomes (for $n = 25$)

$$\epsilon = \frac{1.8125}{3.7 \text{ X } 10^{16}} \ll 10^{-9}$$

This requires a considerably greater amount of computation time since a floating point multiplication is now three "old" floating point multiplications and two floating point additions, however even for $n = 60$, $\epsilon < 10^{-9}$. For comparative purposes, note Table III-2. The choice of either packed or double precision is really dependent on the order of the matrix expected and the requirements of the problem, it seems to this writer that both procedures should be available.

It must be mentioned in passing that the method of Hotelling and Bodewig[1] for iterating to find an improved inverse from an approximate

[1] A. S. Householder, Principles of Numerical Analysis, McGraw-Hill, New York, 1953, pp. 80 ff.

inverse and the original matrix, is readily available. However, although the programming of this scheme would be extremely simple, inasmuch as the basic matrix multiplication is already present, two n^3 multiplications are required for each successive improvement, and furthermore, another n x n matrix must be stored. These requirements with regard to storage and computation time make the double precision floating point arithmetic routine more attractive, since the time to solve an n x n and perform one iteration would be proportional to $\frac{5n^3 + 4n^2 + n}{2}$ X 200 for the packed floating point, whereas with equal storage space, the double precision would only be proportional to $\frac{n^3 + 4n^2 + n}{2}$ X 425.

The decision as to whether or not the simple packed floating point routine or the double precision was to be used would be made in the initial stages of the problem; i.e., when "n + r" was determined, assuming the number of elements was not specified (see Appendix I).

In order to obtain a practical estimate of the accuracy of the matrix inversion program and the time required, the following tests were made:

A symmetric 10 x 10 matrix was formed from a column of random numbers, the sign and position of the decimal point were also established using random numbers. This matrix was then inverted using the program. The matrix product of the original matrix and the inversion was then compared with a unit matrix. The rms. error of each element was found to be 431×10^{-10}, the maximum error was 2430×10^{-10}.

A similar test for 6 x 6 matrix resulted in an rms. error of 14×10^{-9} and a maximum error of 43×10^{-9}.

The running time was determined to be approximately $.064\ n^3$ minutes, (64 minutes for a 10 x 10, 14 minutes for a 6 x 6).

The flow chart and program for the elimination form of the inverse using simple packed floating point airthmetic is appended hereto.

TABLE III-1

COMPARISON OF VARIOUS METHODS

Method	No. of Operations	Word-Times (n=20)*
Stiefel-Hestenes[1]	$2n^3 + ...$	112×10^4
Strict Inversion[1]	$2n^3 + n^2$	115×10^4
Orthogonalization[1]	$2n^3 + n^2/2$	113×10^4
Elimination Form[2]	$1/2\ (n^3 + 2n^2 + n)$	31×10^4
Back-Substitution	$1/6\ (2n^3 + 5n^2$	21×10^4

*Assumes product operation requires 70 word-times.

TABLE III-2

COMPUTATION TIME IN WORD-TIMES

Operation	Normal Float	Packed Float	Double Precision Float
Multiply	80	200	425
Divide*	112	232	684

*Assuming overflow occurs 1/4 of the time.

[1] Householder, (9)

[2] Von Neumann, (24)

MATRIX INVERSION PROGRAM (CRC - 102A)

ADD	INST	M^1	M^2	M^3	REMARKS
0100	30	0076	2100	0371	Store N
0101	30	0077	2100	0366	Store (n + r)
0102	30	0371	0361	0355	Shift N to M^1 position
0103	05	3000	3000	0370	Clear cells
0104	34	3000	2100	2000	0500 - 1777
0105	35	0377	2100	0106	Initialize 0106
0106	30	0077	2100	1777	Shift the
0107	35	0106	0376	0106	contents of Channel 0
0110	34	0375	0106	0106	to Channel 17
0111	35	2100	2100	0373	Set RC (row count) to 0
0112	35	0371	0335	2006	N + 1 modifier
0113	32	0337	0332	0114	Initialize 0114
0114	30	0367	2100	0514	Build
0115	35	0114	2006	0114	N x N
0116	35	0373	0335	0373	unit
0117	34	0371	0373	0114	matrix
0120	30	0335	2100	0336	Set CM (column count) to 1
0121	35	0334	0366	0365	1701 + (n + r) = S
0122	34	3000	2100	0124	Skip 0123 initially
0123	35	0336	0335	0336	Increment CM by 1
0124	32	2100	0332	0152	Set putaway to 0000
0125	35	2100	2100	0373	Set RC to 0
0126	32	0362	0363	0135	Reset pickup for a_{ij}
0127	36	0365	0336	0327	S - CM = RS(row selector)
0130	26	2100	2100	0323	Clear CC (column tally) and TS
0131	34	0327	0326	0133	If RS greater 1677, skip to 133
0132	34	3000	2100	0145	No, skip to 0145
0133	30	0327	0361	2006	Shift RS to M^1 position
0134	32	2006	0363	0140	Initalize 0140 to RS
0135	30	0510	2100	2000	Pickup a_{ij} to "A"
0136	34	2000	2100	0140	If A $\neq$ 0, skip to 0140
0137	34	3000	2100	0145	If A = 0, skip to 0145
0140	30	1704	2100	2001	Load column element
0141	34	3400	2100	0400	Transfer to multiply
0142	30	0364	2100	2001	Load TS
0143	34	3000	2100	0400	Transfer to add
0144	30	2000	2100	0364	Store sum in TS
0145	35	0323	0335	0323	Increment CC by 1
0146	35	0135	0360	0135	Increment a_{ij} pickup by 1
0147	35	0327	0335	0327	Increment RS by 1
0150	34	0371	0323	0131	If N greater than CC, repeat
0151	36	0135	0360	0135	No, compensate pickup
0152	30	0364	2100	0000	Putaway row-column product
0153	35	0373	0335	0373	Add 1 to RC
0154	34	0371	0373	0156	If N greater than RC, skip 0155
0155	34	3000	2100	0161	No, skip to 0155, thence to 0161
0156	35	0135	0360	0135	Add 1 to a_{ij} pickup
0157	35	0152	0335	0152	Increment putaway address
0160	34	3000	2100	0127	Repeat loop

MATRIX INVERSION PROGRAM (CRC - 102A)

ADD	INST	M^1	M^2	M^3	REMARKS
0161	36	0336	0335	0357	CM - 1 = J
0162	26	0357	0371	2006	J x N
0163	35	2006	0337	2007	0500 + (J x N)
0164	35	2100	2100	0323	Clear CC
0165	30	2007	0330	0356	Initialize I_{r_1}
0166	32	0356	0363	0172	in 0172
0167	30	0357	0330	2006	Initialize P_{k_r}
0170	32	2006	0363	0173	in 0173
0171	32	2007	0332	0175	Initialize I'_{r_1} in 0175
0172	30	0510	2100	2000	Load I_r
0173	30	0002	2100	2001	Load P_k
0174	34	3600	2100	0400	$I_r \div P_k = I'_r$
0175	30	2000	2100	0510	Putaway result
0176	35	0323	0335	0323	Increment CC by 1
0177	34	0336	0323	0201	If CM greater than CC, 0201
0200	34	3000	2100	0204	If CM = CC, 0204
0201	35	0172	0360	0172	Increment I_{r_j} to $I_{r_{i+1}}$
0202	35	0175	0335	0175	Increment putaway
0203	34	3000	2100	0172	Continue dividing row 'r'
0204	35	2100	2100	0325	Set M to 0
0205	34	0336	0325	0207	If CM greater than M, 0207
0206	34	3000	2100	0242	If CM = M, 0242
0207	35	0337	0325	0322	Build I'_{ij}
0210	30	0322	0330	0324	Build I_{ij}
0211	35	0371	0335	0327	N + 1 to TS
0212	30	0335	2100	0373	Initialize RC to 1
0213	32	0356	0363	0222	Initialize I_{1r} pickup
0214	32	2100	0363	0223	Initialize P_{k_r} pickup
0215	32	0324	0363	0226	Initialize I'_{ij} pickup
0216	32	0322	0332	0230	Initialize I_{ij} pickup
0217	34	0373	0336	0222	If RC $\neq$ CM
0220	34	0336	0373	0222	skip to 0222
0221	34	3000	2100	0231	If RC = CM, skip to 0231
0222	30	0510	2100	2000	Load I_{1r}
0223	30	0002	2100	2001	Load P_{k_r}
0224	34	3400	2100	0400	Multiply
0225	30	2000	2100	2001	Shift for subtraction
0226	30	0510	2100	2000	Load I_{1j}
0227	34	3200	2100	0400	$I'_{1j} = I_{1j} - I_{1r} \times P_{k_r}$
0230	30	2000	2100	0510	Putaway
0231	35	0373	0335	0373	Increment RC by 1
0232	34	0327	0373	0236	If N + 1 greater RC, 0236
0233	35	0356	0360	0356	Increment I_{1r} to I_{2r}
0234	35	0325	0335	0325	Increment M by 1
0235	34	3000	2100	0205	Repeat loop
0236	35	0223	0360	0223	Increment P_{k_r} to $P_{k_{r+1}}$
0237	35	0226	0355	0226	$I_{1j} + N = I_{2j}$
0240	35	0230	0371	0230	Increment putaway
0241	34	3000	2100	0217	Repeat loop
0242	34	0371	0336	0123	Next iteration (α loop)

MATRIX INVERSION PROGRAM (CRC - 102A)

ADD	INST	M^1	M^2	M^3	REMARKS
0243	35	2100	2100	0336	Set CM = 0
0244	35	0347	2100	0152	Reset putaway
0245	30	0367	2100	0000	Putaway "1" in 0000
0246	32	0376	0332	0152	Dummy (see 0244)
0247	35	0354	2100	0155	Mod 0155 ("out comd")
0250	34	3000	2100	0125	Transfer for N+1st column
0251	35	0346	2100	0152	Reset putaway
0252	35	0353	2100	0155	Restore 0155
0253	35	2100	2100	0352	Set EC (equation count) = 0
0254	32	0351	0332	0276	Initialize putaway
0255	34	3000	2100	0260	Skip modifiers
0256	35	0352	0435	0352	Increment EC by 1
0257	35	0276	0435	0276	Increment putaway
0260	32	0351	0363	0267	Reset h_0
0261	35	2100	2100	0364	Clear TS
0262	30	0352	0361	2006	Shift EC
0263	32	2006	0363	0270	Extract into b pickup
0264	34	3000	2100	0267	Jump modifiers
0265	35	0267	0360	0267	Increment h
0266	36	0270	0360	0270	Decrement b
0267	30	1701	2100	2000	Load h
0270	30	0000	2100	2001	Load b
0271	34	3400	2100	0400	Multiply
0272	30	0364	2100	2001	Load TS
0273	34	3000	2100	0400	Add
0274	30	2000	2100	0364	Putaway summation
0275	34	0270	0350	0265	If b greater than 0, repeat
0276	30	0364	2100	1001	No, summation to answer storage
0277	34	0366	0352	0256	If (n + r) greater EC, 0256
0300	35	2100	2100	0345	Clear tally, inversion completed
0301 - 0321		Available for boot-strapping routines Denominator in Channel 0 Numerator in Channel 1			
0322	00	0000	0000	0502	I'_{ij} in M^3 storage
0323	00	0000	0000	0003	CC storage
0324	00	0502	0000	0000	I'_{ij} in M^1 storage
0325	00	0000	0000	0003	M storage
0326	00	0000	0000	1677	Testword for 0131
0327	00	0000	0000	1705	RS storage
0330	00	0000	0000	0030	Shift Control
0331	35	2000	2005	2000	Clear Routine (2001)
0332	00	0000	0000	3777	M^3 extractor
0333	34	3000	2100	0105	Clear Routine (2003)
0334	00	0000	0000	1701	Constant for 0116
0335	00	0000	0000	0001	Constant for Clear Routine
0336	00	0000	0000	0000	CM storage
0337	00	0000	0000	0500	Constant for 0113 etc.

MATRIX INVERSION PROGRAM (CRC - 102A)

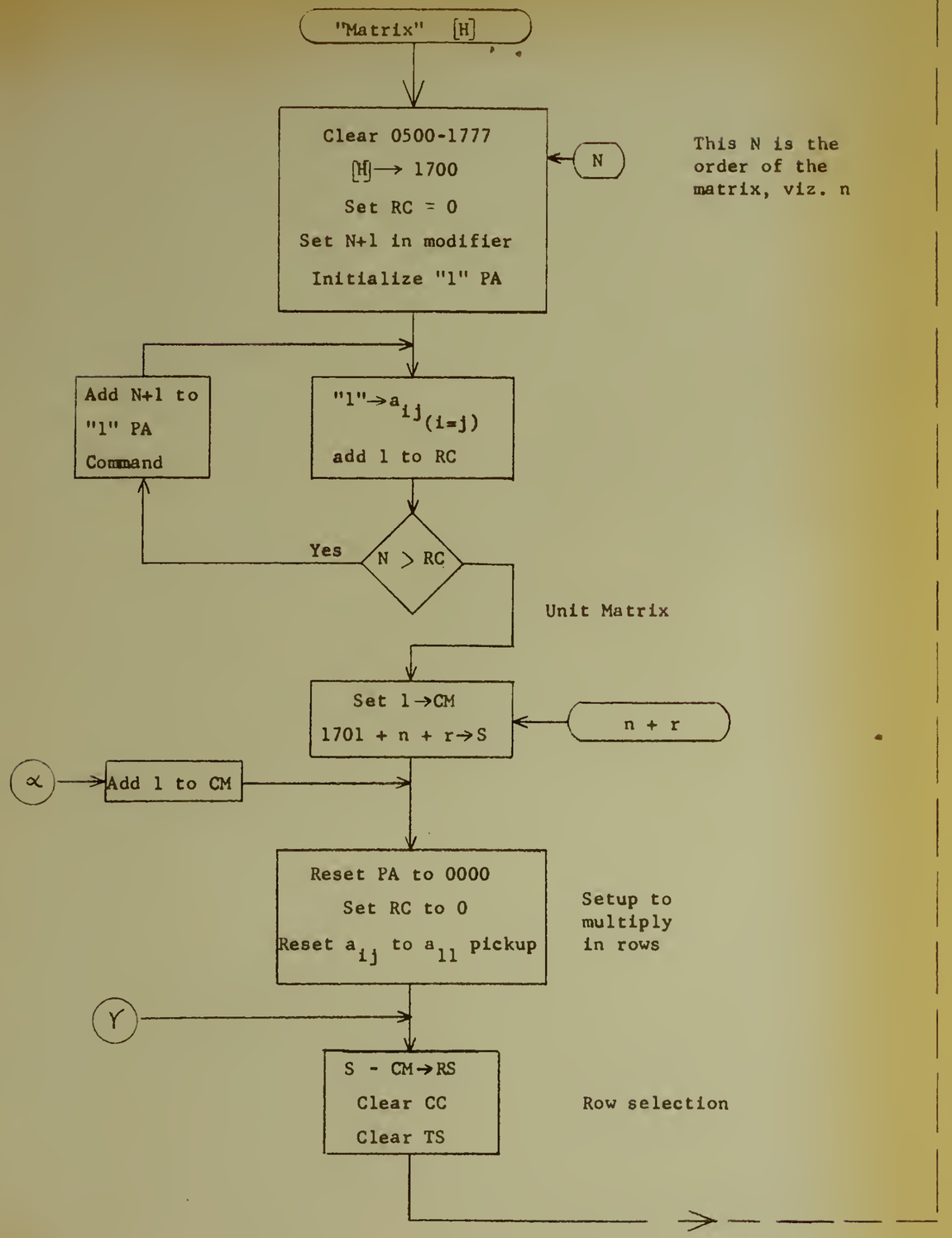
ADD	INST	M^1	M^2	M^3	REMARKS
0340	00	0000	0000	0000	Available
0341	00	1000	0000	0000	Constant for 0156
0342	10	0100	0000	0000	Not required
0343	00	0000	0000	0004	Not required
0344	00	0000	0000	0004	Not required
0345	00	0000	0000	0004	Not required
0346	30	0364	2100	0000	Reset for 0152
0347	36	2100	2000	0000	Reset for 0244
0350	00	0000	2100	2001	Testword for 0275
0351	00	1700	0000	1000	Putaway reset
0352	00	0000	0000	0001	EC tally
0353	34	3000	2100	0161	Modifier for 0252
0354	34	3000	2100	0251	Modifier for 0247
0355	00	0003	0000	0000	N in M^1
0356	00	0511	0000	0000	I_{r1} in M^1
0357	00	0000	0000	0002	J tally
0360	00	0001	0000	0000	Constant for 0146
0361	00	0000	0000	0030	Shift control word
0362	00	0500	0000	0000	Constant for 0126
0363	00	7777	0000	0000	M^1 extractor
0364	02	2314	1540	0207	Temporary storage (TS)
0365	00	0000	0000	1702	S storage
0366	00	0000	0000	0001	(n + r) storage
0367	00	4000	0000	0001	"1" in PackBinFltPt
0370	26	2100	2100	0500	Clear Routine (2000)
0371	00	0000	0000	0003	N storage
0372	34	2004	2000	2000	Clear Routine (2002)
0373	00	0000	0000	0003	RC tally
0374	00	2100	2100	1400	Clear Routine (2004)
0375	00	0077	2100	1777	Testword for 0105
0376	00	0001	0000	0001	Constant for 0104
0377	30	0000	2100	1700	Not required

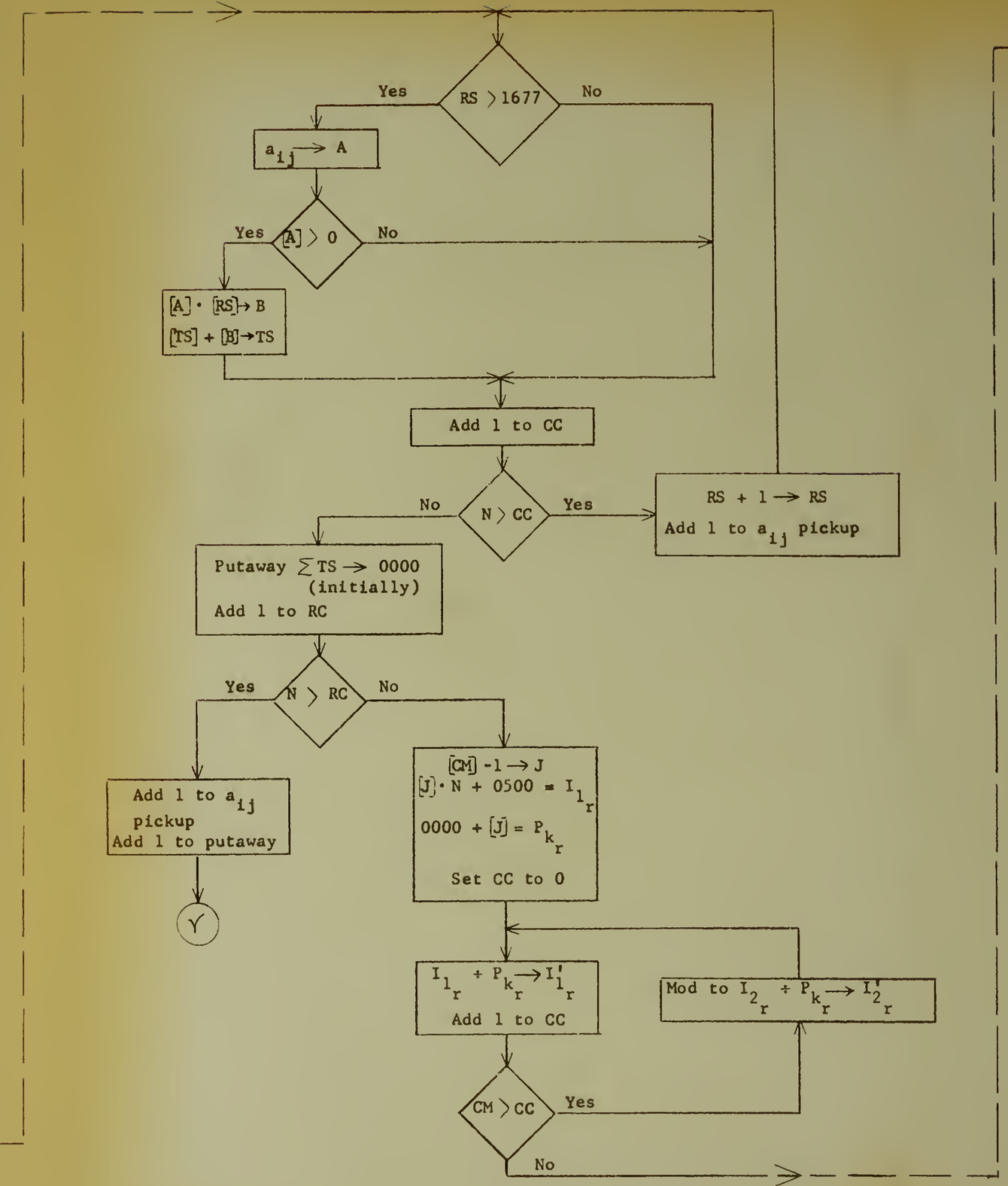
Channel 4 contains packed floating point arithmetic routines

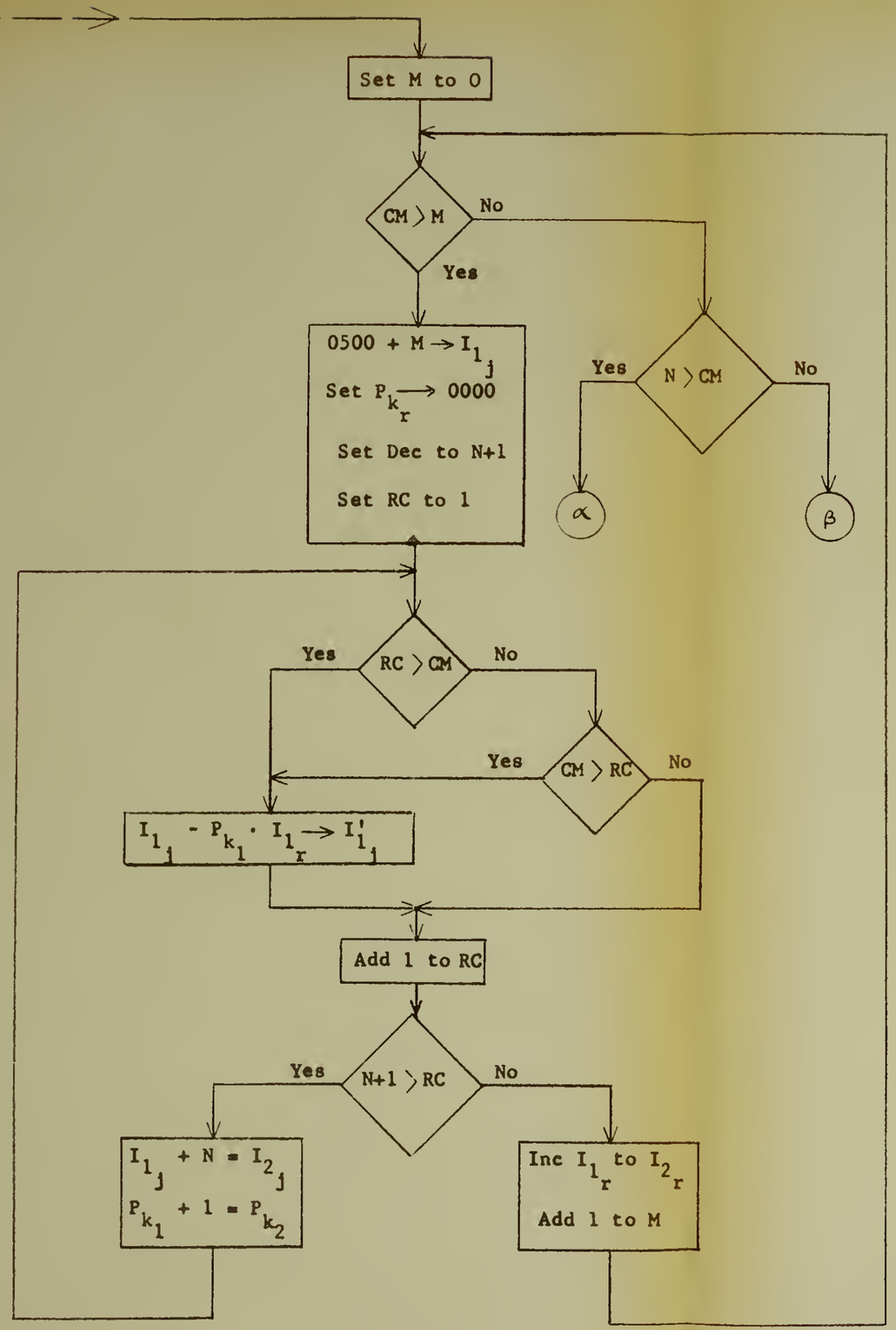
0400	30	3000	0443	2006	Prepare
0401	32	2006	0475	0433	exit
0402	30	2001	2100	2007	Store 2nd operand
0403	32	2000	0434	2102	Extract sign of exp^1
0404	27	2002	0451	2002	Shift to operating location
0405	32	2000	0453	2002	Extract mag of exp^1
0406	32	2000	0444	2103	Extract sign and mag of mag^1
0407	31	2002	2003	2000	Shift to 2000,2001
0410	32	2007	0434	2102	Extract sign of exp^2
0411	27	2002	0451	2002	Shift to operating location
0412	32	2007	0436	2002	Extract mag of exp^2
0413	32	2007	0444	2103	Extract sign and mag of mag^2
0414	36	2006	0435	2006	Pickup entry command

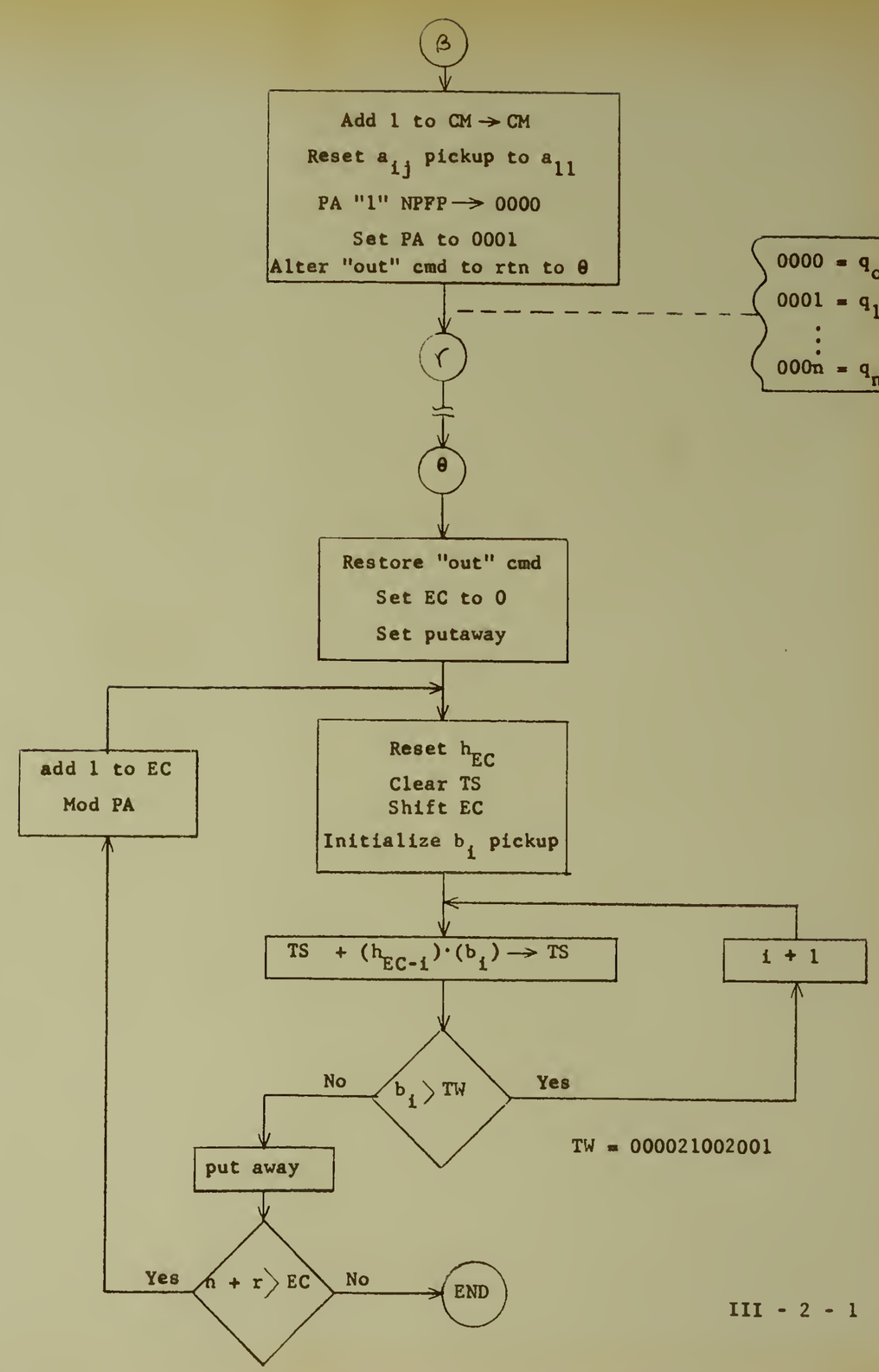
ADD	INST	M^1	M^2	M^3	REMARKS
0415	30	2006	0437	2006	Shift to M^1
0416	32	2006	0461	0417	Set 0417
0417	32	0273	0461	2107	Extract control
0420	34	2007	0400	0440	If control ≥ 3000, 0440
0421	33	2002	2000	0467	No, operations
0422	36	2002	2000	2006	for normal
0423	30	2003	2006	2007	floating point
0424	35	2001	2007	2001	addition
0425	37	2001	3000	0462	routine
0426	31	2000	2001	2002	Store in 2002, 2003
0427	27	2002	0452	2104	Shift sign of exp for pack
0430	32	2002	0436	2004	Extract mag of exp for pack
0431	32	2004	0476	2003	Pack exponent
0432	30	2003	2100	2000	Packed result to 2000
0433	34	3000	2100	0274	Exit to main program
0434	00	0000	0000	0100	Extractor for sign of exp
0435	00	0000	0000	0001	Constant for 0414
0436	00	0000	0000	0077	Extractor for mag of exp
0437	00	0000	0000	0030	Shift control word
0440	34	2007	0450	0445	If control ≥ 3200, 0445
0441	36	2100	2003	2003	Change sign of 2^{nd} operand
0442	34	3000	2100	0421	Transfer to addition
0443	02	0000	0000	0030	Shift control word
0444	02	7777	7777	7600	Extractor for magnitude
0445	34	2007	0474	0454	If control ≥ 3400, 0454
0446	35	2000	2002	2000	No, operation for normal
0447	25	2001	2003	2001	floating point
0450	34	3200	2100	0426	multiplication
0451	00	0000	0000	0037	Shift to pack commands
0452	02	0000	0000	0037	Shifter for sign of exp
0453	00	0000	0000	0077	Extractor for mag of exp
0454	36	2000	2002	2000	Operations for
0455	23	2001	2003	2001	normal floating point
0456	37	2001	3000	0462	division
0457	34	3000	2100	0426	Transfer to pack commands
0460	02	0000	0000	0001	Constant for 0462
0461	00	3777	0000	0000	M^1 extracotor
0462	27	2001	0460	2001	Normal
0463	30	2001	0460	2001	correct
0464	27	2001	0477	2001	overflow
0465	35	2000	0477	2000	procedure
0466	34	3000	2100	0426	Transfer to pack commands
0467	36	2000	2002	2006	Operations for
0470	30	2001	2006	2007	normal floating
0471	30	2002	2100	2000	point addition
0472	35	2003	2007	2001	when 2002 ≥ 2000
0473	37	2001	3000	0462	see 0421
0474	34	3400	2100	0426	Transfer to pack commands
0475	00	0000	0000	7777	M^3 extractor
0476	00	0000	0000	0177	Sign and exponent extractor
0477	00	0000	0000	0001	Constant

FLOW CHART OF MATRIX INVERSION AND SOLUTION






Set M to 0
CM > M
No
Yes
0500 + M → I1j
Set Pkr → 0000
Set Dec to N+1
Set RC to 1
Yes
N > CM
No
α
β
Yes
RC > CM
No
Yes
CM > RC
No
I1i - Pk1 · I1r → I'1j
Add 1 to RC
Yes
N+1 > RC
No
I1j + N = I2j
Pk1 + 1 = Pk2
Inc I1r to I2r
Add 1 to M


β
Add 1 to CM → CM
Reset aij pickup to a11
PA "1" NPFP → 0000
Set PA to 0001
Alter "out" cmd to rtn to θ
0000 = q0
0001 = q1
000n = qn
γ
θ
Restore "out" cmd
Set EC to 0
Set putaway
add 1 to EC
Mod PA
Reset hEC
Clear TS
Shift EC
Initialize bi pickup
TS + (hEC-i)·(bi) → TS
i + 1
No
bi > TW
Yes
TW = 000021002001
put away
Yes
n + r > EC
No
END

APPENDIX IV

LAGRANGIAN INTERPOLATION METHOD FOR THE SOLUTION OF ALGEBRAIC EQUATIONS[1]

The Lagrangian interpolation method described by D. E. Muller was selected for the solution of large degree polynominals. This procedure was programmed, with certain modifications, and tested on the CRC-102A computer. By removing the roots, as found, by synthetic division, rather than by the algorithm recommended by Muller, accuracies of at least eight decimal places were obtained for all roots of equations up to degree 21. This was considerably better than the accuracies reported by Muller, and incidentally, probably entailed considerably more iteration.

A graph of solution time vs. degree of equation, for $\epsilon = 10^{-13}$, is presented in Figure IV-1. Convergence to a root generally requires between 7 and 14 iterations (for the ϵ of 10^{-13}). The iterative loop is completed in about 1.5 minutes, varying slightly (not more than 10%) according to the degree of the equation. In making these tests, it was found that when equations had roots of the same modulus, there is a pronounced slow down in the rate of convergence. The equations $x^{24} + 1 = 0$, and $x^{20} + 1 = 0$ required about 12.5 minutes per root, whereas the more general polynominal, (equation A under Results of Tests) only required about 9.4 minutes per root. Round-off error was not present in the $x^{20} + 1$ equation and affected only two roots in the solution of $x^{24} + 1 = 0$ within the eighth decimal place.

[1] D. E. Muller, Solving Algebraic Equation by an Automatic Computer, Mathematical Tables and Other Aids to Computation, Vol. X, No. 56, Oct., 1956, pp. 208-215.

Thus far the program has failed only once. When solving an equation with a real double negative root and a pair of complex roots of about the same modulus, viz., 4; the rate of convergence was normal until about four decimal places had been established and decreased so as to become barely perceptible. The program did, however, remove all roots interior to modulo 4 without difficulty.

The basic outline of the Lagrangian interpolation method is as follows:

Successive iterations toward any particular root are obtained by finding the nearer root of a quadratic whose curve passes through the points; z_i, $f(z_i)$; z_{i-1}, $f(z_{i-1})$; z_{i-2}, $f(z_{i-2})$.

The Lagrangian formula may be written as $L_i\left[f(z)\right] = b_0z^2 + b_1z + b_2$; where the b's satisfy the system of equations:

$$b_0z_i^2 + b_1z_i + b_2 = f(z_i)$$

$$b_0z_{i-1}^2 + b_1z_{i-1} + b_2 = f(z_{i-1})$$

$$b_0z_{i-2}^2 + b_1z_{i-2} + b_2 = f(z_{i-2})$$

and these equations, of course, become almost one in the same as the iterative process converges, depending on the accuracy which we specify.

We are to construct a polynominal (quadratic) whose curve passes through the three points. By the Lagrangian interpolation formula[2], we write:

$$L_i\left[f(z)\right] = f(z_i)\frac{(z - z_{i-1})(z - z_{i-2})}{(z_i - z_{i-1})(z_i - z_{i-2})} + f(z_{i-1})\frac{(z - z_i)(z_i - z_{i-2})}{(z_{i-1} - z_i)(z_{i-1} - z_{i-2})}$$

$$+ f(z_{i-2})\frac{(z - z_i)(z - z_{i-1})}{(z_{i-2} - z_i)(z_{i-2} - z_{i-1})}$$

[2]W. E. Milne, Numerical Calculus, Princeton University Press, Princeton, N. J., 1949, for example.

If we make the following simplifying substitutions:

$$z - z_i = h \qquad h/h_i = \lambda$$

$$z_i - z_{i-1} = h_i \qquad h_i/h_{i-1} = \lambda_i$$

$$z_{i-1} - z_{i-2} = h_{i-1} \qquad 1 + \lambda_i = \Delta_i$$

we may express the coefficient of $f(z_i)$ as:

$$\lambda^2 \Delta_i^{-1} {}_i + \lambda \Delta_i^{-1}(\lambda + \Delta_i) + 1$$

since $$\frac{(z - z_{i-1})(z - z_{i-2})}{(z_i - z_{i-1})(z_i - z_{i-2})} = \left[\lambda + 1\right] \frac{(z - z_{i-2})}{(z_i - z_{i-2})}$$

$$\frac{1}{\lambda_i} = \frac{z_{i-1} - z_{i-2}}{z_i - z_{i-1}}, \text{ and}$$

$$1 + \lambda + \frac{1}{\lambda_i} = \frac{z - z_{i-2}}{z_i - z_{i-1}} = \frac{\Delta_i}{\lambda_i} + \lambda$$

$$\lambda_i \left[\frac{\Delta_i}{\lambda_i} + \lambda\right] = \frac{z - z_{i-2}}{z_{i-1} - z_{i-2}} \quad \text{and finally}$$

$$\Delta^{-1} \lambda_i \left[\frac{\Delta_i}{\lambda_i} + \lambda\right] = \frac{z - z_{i-2}}{z_i - z_{i-2}} \quad ; \text{ therefore,}$$

$$(1 + \lambda)(1 + \lambda \lambda_i \Delta_i^{-1}) = \frac{(z - z_{i-1})(z - z_{i-2})}{(z_i - z_{i-1})(z_i - z_{i-2})} \quad \text{the coefficient of } f(z_i),$$

which when expanded is:

$$\lambda^2 \lambda_i \Delta_i^{-1} + \lambda \lambda_i \Delta_i^{-1} + \lambda + 1 = \lambda^2 \lambda_i \Delta_i^{-1} + \lambda \Delta_i^{-1}(\lambda_i + \Delta_i) + 1 \text{ (QED).}$$

By similar manipulation, the coefficients of $f(z_{i-1})$ and $f(z_{i-2})$ can be found and the desired quadratic in λ is:

$$L_i\left[f(z)\right] = \lambda^2 \Delta_i^{-1}\left[f(z_{i-2})\,\lambda_i^2 - f(z_{i-1})\,\lambda_i\Delta_i + f(z_i)\lambda_i\right] +$$

$$\lambda\Delta^{-1}\left[f(z_{i-2})\lambda_i^2 - f(z_{i-1})\,\Delta_i^2 + fz_i(\lambda_i + \Delta_i)\right] + f(z_i)$$

A single iterative step is obtained by letting z_{i+1} be such a value of z that $L_i\left[f(z)\right] = 0$. We may solve the quadratic inversely to obtain λ which is $\lambda_{i+1} = \frac{z_{i+1} - z_i}{z_i - z_{i-1}}$ from which we obtain the new z_i and repeat the iterative process.

All operations are in complex numbers and in floating point arithmetic. A root is taken when

$$\left|\frac{z_{i+1} - z_i}{z_i}\right| \leqslant \epsilon$$

For proof of convergence of the process the reader is referred to Dr. Muller's paper.

RESULTS OF TESTS

Equation A

$$z^{16} - 9.0\, z^{15} + 34.75\, z^{14} - 102.25\, z^{13} + 264.0\, z^{12} - 479.0\, z^{11}$$

$$+ 771.5\, z^{10} - 1154.5\, z^{9} + 978.0\, z^{8} - 1225.0\, z^{7} + 912.75\, z^{6}$$

$$+ 254.75\, z^{5} + 1411.0\, z^{4} + 1319.0\, z^{3} + 1022.5\, z^{2} + 470.5\, z$$

$$+ 105.0 \quad \epsilon = 2^{-44} (= 10^{-13})$$

Total solution time: 2 hours, 34 minutes, 10 seconds on CRC-102A.

Roots:

-0.284460054	± j	0.598759363
-0.376766336	± j	0.188840569
-0.101279181	± j	1.429775042
-0.044975589	± j	1.738162299
+1.999999996		
-0.011258189	± j	1.933289317
+2.500000009		
-0.181260648	± j	1.034830851
+3.000000001		
+3.499999991		

Equation B

$x^{30} + 1 = 0$ $\epsilon = 2^{-44}$ $(= 10^{-13})$

Total solution time: 6 hours, 23 minutes, 30 seconds on CRC-102A

Roots:

-0.000000000	± j	0.999999999
-0.951056516	± j	0.309016994
+0.866025403	± j	0.500000000
-0.587785252	± j	0.809016994
+0.994521895	± j	0.104528463
-0.207911690	± j	0.978147601
-0.994521895	± j	0.104528463
+0.406736637	± j	0.913545463
+0.207911688	± j	0.978147598
+0.743144829	± j	0.669130603
-0.743144826	± j	0.669130607
+0.587785258	± j	0.809017000
-0.866025404	± j	0.499999998
+0.951056517	± j	0.309016993
-0.406736643	± j	0.913545458

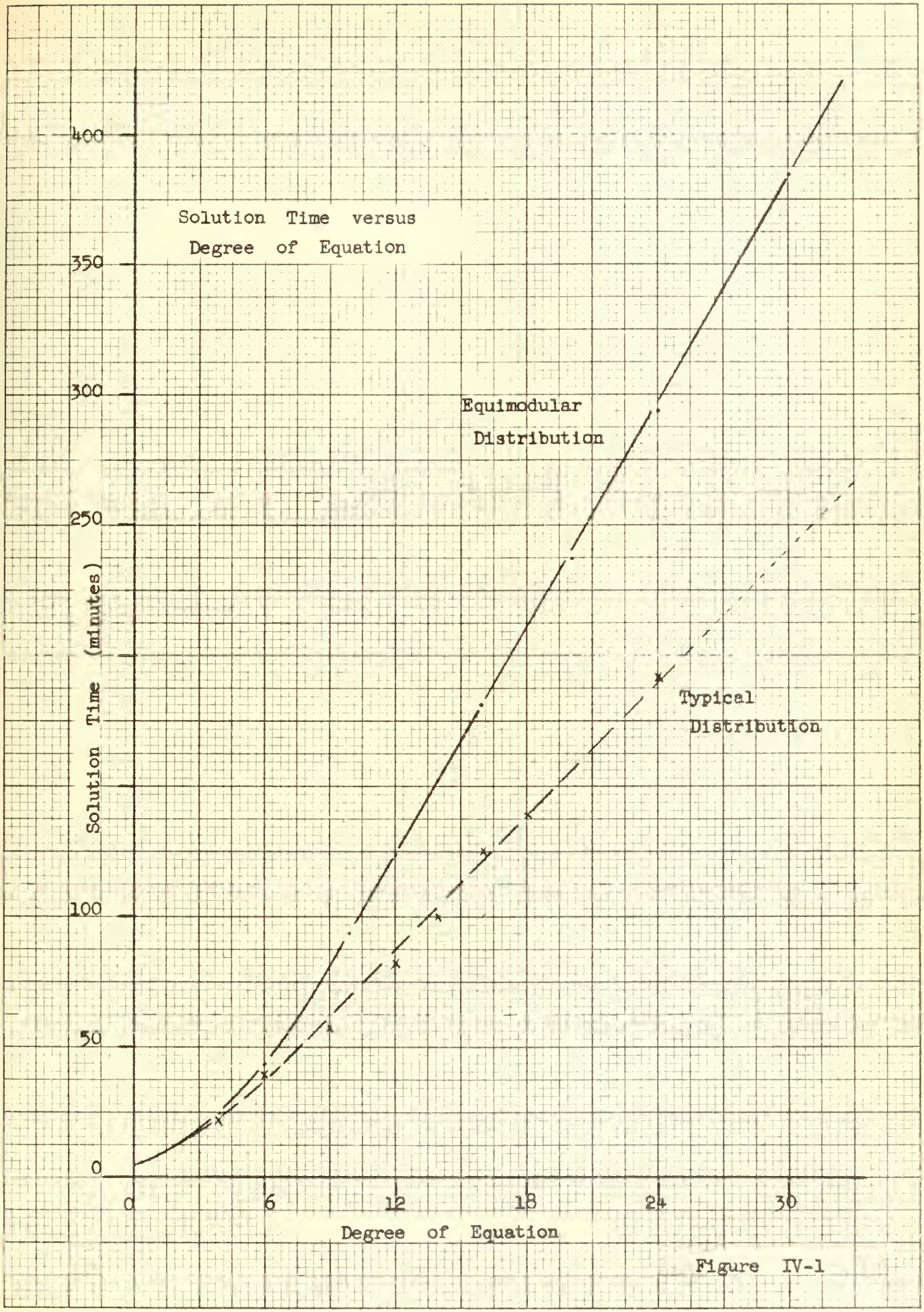

Solution Time versus
Degree of Equation
Equimodular
Distribution
Typical
Distribution
Solution Time (minutes)
400
350
300
250
100
50
0
0
6
12
18
24
30
Degree of Equation

Figure IV-1

LAGRANGIAN INTERPOLATION METHOD FOR HIGH DEGREE POLYNOMIALS

ADD	INST	M^1	M^2	M^3	REMARKS
Channels 0 and 1 are used temporary storage.					
Channel 2 contains a decimal to binary floating point conversion which while not needed in overall program is included here.					
0200	26	1340	0250	2005	Build and
0201	35	0246	2005	0233	store testword
0202	35	0251	2100	0210	Reset pickup
0203	35	0253	2100	0211	Reset pickup
0204	32	2100	0252	0270	Reset putaway
0205	05	3000	3000	3000	Clear buffer
0206	30	0254	2100	0265	Reset extractor
0207	32	0254	0255	2004	Set 2004 to 1
0210	30	0016	2100	0230	Pickup integer
0211	30	0017	2100	0247	Pickup fraction
0212	34	0230	2100	0214	Is integer $\succ$ 0?
0213	34	3000	2100	0231	No, test fraction
0214	32	0230	0255	0227	Yes, extract sign
0215	32	0230	0265	2005	Extract least sig. digit
0216	26	2004	2005	2006	L.S.D. x (2004)
0217	35	2006	2001	2001	Accumulate products
0220	26	2004	0257	2004	Increase 2004 by power of 10
0221	30	0230	0262	0230	Shift decimal word 4 right
0222	34	0230	2100	0215	If conversion not complete, 0215
0223	31	2000	2001	2000	If complete normalize
0224	35	2000	0261	2000	and correct exponent
0225	34	0247	2100	0235	Is fraction $\succ$ 0?
0226	34	3000	2100	0267	No, skip to putaway
0227	00	0000	0000	0000	Sign storage
0230	00	0000	0000	0000	Integer storage
0231	34	0247	2100	0234	I = 0, Is fraction $\succ$ 0?
0232	34	3000	2100	0267	If both are 0, putaway 0
0233	00	0016	2100	0230	Testword
0234	32	0247	0255	0227	Extract sign when fraction only
0235	32	0247	0265	2003	Extract least sig. frac. dec. dig.
0236	23	2003	0263	2003	Divide by 10
0237	30	0265	0256	0265	Shift extractor
0240	32	0247	0265	2003	Extract next digit
0241	23	2003	0263	2003	Divide by 10
0242	34	0260	0265	0237	If conversion not complete, 0237
0243	34	2001	2100	0266	If integer $\succ$ 0, transfer to add
0244	31	2002	2003	2000	If integer was 0, normalize
0245	34	3000	2100	0267	and putaway
0246	00	0002	2100	0230	Testword addend
0247	00	0000	0000	0000	Fraction storage
0250	00	0002	0000	0000	Testword build constant
0251	30	0000	2100	0230	Integer pickup reset
0252	00	0000	0000	7777	M^3 extractor

LAGRANGIAN INTERPOLATION METHOD FOR HIGH DEGREE POLYNOMIALS

ADD	INST	M^1	M^2	M^3	REMARKS
0253	30	0001	2100	0247	Fraction pickup reset
0254	00	0000	0000	0017	Extractor reset
0255	02	0000	0000	0001	Sign extractor and -1
0256	00	0000	0000	0004	Shift control - 4 left
0257	00	0000	0000	0012	"Power of 10"
0260	00	7400	0000	0000	Extractor testword
0261	00	0000	0000	0044	To correct exponent
0262	02	0000	0000	0004	Shift control - 4 right
0263	00	5000	0000	0000	10 for division
0264	00	0000	0000	0002	2
0265	00	7400	0000	0000	Extractor working storage
0266	34	3000	2100	1540	Transfer to addition
0267	32	0227	0255	2001	Extract sign into mag portion
0270	30	2000	2100	0016	Putaway exponent
0271	36	0270	0255	2007	Build magnitude
0272	32	2007	0252	0273	putaway
0273	30	2001	2100	0015	Putaway magnitude
0274	35	0210	0250	0210	Modify integer pickup
0275	35	0211	0250	0211	Modify fraction pickup
0276	35	0270	0264	0270	Modify putaway
0277	34	0233	0210	0205	Have all numbers been converted?

Channels 3 through 7, and 1000 through 1012 contain main program for Lagrangian Interpolation Method; in the program detailed here Channels 10 and 11 contain a binary floating point to decimal conversion routine which would not be required for the overall program. Channel 13 is used for tempoary storage and Channels 14 through 16 contain the floating point arithmetic routines for complex numbers.

LAGRANGIAN INTERPOLATION METHOD FOR HIGH DEGREE POLYNOMIALS

ADD	INST	M^1	M^2	M^3	REMARKS
0300	30	1340	2100	1372	N to N'
0301	05	3000	3000	0000	Transfer
0302	04	3000	3000	0100	the contents
0303	35	0301	1376	0301	of Channel 0
0304	35	0302	1376	0302	to
0305	34	1374	0302	0301	Channel 1
0306	36	0301	1437	0301	and
0307	36	0302	1437	0302	reset.
0310	30	1331	2100	2000	
0311	36	2100	1337	2001	Initial setup
0312	31	2000	2001	1300	for z_{i-2}
0313	31	1333	1337	1320	
0314	31	1331	1337	1301	Initial setup
0315	31	1333	1337	1321	for z_{i-1}
0316	31	1333	1337	1302	Initial setup
0317	31	1333	1337	1322	for z_i
0320	35	2100	2100	1303	Initial setup
0321	36	2100	1337	1344	for λ_i
0322	31	1333	1337	1323	$= 1/2 + j0$
0323	26	1372	1466	2007	Commence pickup build
0324	26	2100	2100	1434	Clear 1434, 1475
0325	26	2100	2100	1435	Clear 1435, 1476
0326	35	0640	2007	0327	Set pickup for A_0
0327	31	0102	0103	2000	Pickup A_0
0330	31	2000	2001	1307	A_0 is Re of $f(z_i)$
0331	31	1333	1337	1327	$\mathrm{Im} f(z_i) = 0$
0332	36	0327	1473	2007	Build A_n pickup
0333	32	2007	1464	0334	and enter in 0334
0334	31	0070	0071	2002	A_n pickup command
0335	34	3000	2100	1540	Transfer to floating addition
0336	36	0334	1473	0334	Modify A_n pickup for A_{n-2}
0337	34	0640	0334	0341	If sum complete, putaway
0340	34	3000	2100	0334	No, repeat to 0334
0341	34	1475	2100	0347	Has even sum putaway been made?
0342	31	2000	2001	1434	No, even sum to 1434, 1475
0343	36	0327	1466	0344	Build A_{n-1} pickup
0344	31	0100	0101	2000	A_{n-1} pickup
0345	36	0344	1473	2007	Set An pickup for odd terms
0346	34	3000	2100	0333	and repeat back for odd sum
0347	31	2000	2001	1435	Putaway odd sum in 1435, 1476
0350	31	1434	1475	2002	Load even sum
0351	34	3000	2100	1540	Odd + even for
0352	31	2000	2001	1306	$f(z_{i-1})$
0353	31	1333	1337	1526	and clear imaginary
0354	31	1434	1475	2000	Load even sum
0355	31	1435	1476	2002	and odd sum
0356	34	3000	2100	1516	for subtraction
0357	31	2000	2001	1305	to form $f(z_{i-2})$

ADD	INST	M^1	M^2	M^3	REMARKS
0360	31	1333	1337	1325	and clear imaginary
0361	31	1303	1344	2000	Load
0362	31	1323	1364	2002	λ_i
0363	31	1331	1335	2004	Load
0364	31	1333	1337	2006	1 + j0
0365	34	3000	2100	1410	Transfer to cpx. add.
0366	31	2000	2001	1304	Putaway
0367	31	2002	2003	1324	Δ_i
0370	31	1306	1347	2004	Load
0371	31	1326	1367	2006	$f(z_{i-1})$
0372	34	3000	2100	1607	Transfer to cpx. mpy.
0373	04	3000	3000	1330	Store in B
0374	31	1305	1346	2000	Load
0375	31	1325	1366	2002	$f(z_{i-2})$
0376	31	1303	1344	2004	Load
0377	31	1323	1364	2006	λ_i
0400	34	3000	2100	1607	Transfer to cpx. mpy.
0401	04	3000	3000	1310	Store in A
0402	31	1307	1350	2004	Load
0403	31	1327	1370	2006	$f(z_i)$
0404	34	3000	2100	1410	$f(z_{i-2})\lambda_i + f(z_i)$
0405	31	1330	1371	2004	Load
0406	31	1332	1373	2006	$f(z_{i-1})\Delta_i$
0407	34	3000	2100	1416	$[f(z_{i-2})\lambda_i = f(z_{i-1})\Delta_i + f(z_i)]$
0410	31	1304	1345	2004	Load
0411	31	1324	1365	2006	Δ_i
0412	34	3000	2100	1607	$f(Z)\cdot\Delta_i$
0413	31	1303	1344	2004	Load
0414	31	1323	1364	2006	λ_i
0415	34	3000	2100	1607	$f(Z)\cdot\Delta_i\cdot\lambda_i$
0416	31	1307	1350	2004	Load
0417	31	1327	1370	2006	$f(z_i)$
0420	35	2004	0264	2004	and "multiply
0421	35	2006	0264	2006	by 4
0422	34	3000	2100	1607	$4f(z_i)\Delta_i\lambda_i$ $f(Z) = 4\cdot AC$
0423	04	3000	3000	1440	Store in C
0424	05	3000	3000	1330	Load $f(z_{i-1})\cdot\Delta_i$
0425	31	1304	1345	2004	Load
0426	31	1324	1365	2006	Δ_i
0427	34	3000	2100	1607	$f(z_{i-1})\cdot\Delta_i^2$
0430	04	3000	3000	1330	Store in B
0431	05	3000	3000	1310	$f(z_{i-2})\cdot\lambda_i$ from A
0432	31	1303	1344	2004	Load λ_i
0433	31	1323	1364	2006	
0434	34	3000	2100	1607	Multiply for $f(z_{i-2})\lambda_i^2$
0435	04	3000	3000	1310	Store in A
0436	31	1303	1344	2000	Load
0437	31	1323	1364	2002	λ_i

LAGRANGIAN INTERPOLATION METHOD FOR HIGH DEGREE POLYNOMIALS

ADD	INST	M^1	M^2	M^3	REMARKS
0440	31	1304	1345	2004	Load
0441	31	1323	1365	2006	Δ_i
0442	34	3000	2100	1410	$(\lambda_i + \Delta_i)$
0443	31	1307	1350	2004	Load
0444	31	1327	1370	2006	$f(z_i)$
0445	34	3000	2100	1607	$f(z_i)(\lambda_i + \Delta_i)$
0446	31	1310	1351	2004	Load
0447	31	1312	1353	2006	$f(z_{i-2})\lambda_i^2$
0450	34	3000	2100	1410	$f(z_i)(\lambda_i + \Delta_i) + f(z_{i-2})\lambda_i^2$
0451	31	1330	1371	2004	Load
0452	31	1332	1373	2006	$f(z_{i-1})\Delta_i^2$
0453	34	3000	2100	1416	$f(z_{i-2})\lambda_i^2 - f(z_{i-1})\Delta_i^2 + f(z_i)(\lambda_i+\Delta_i)$
0454	04	3000	3000	1330	Result = g_i; store in B
0455	31	2000	2001	2004	Load B for complex
0456	31	2002	2003	2006	multiplication, i.e. B^2
0457	34	3000	2100	1607	g_i^2 OR B^2
0460	31	1440	1401	2004	Load 4AC
0461	31	1442	1403	2006	$B^2 - 4AC$
0462	34	3000	2100	1416	Store in C
0463	04	3000	3000	1440	Load
0464	31	2000	2001	2004	$B^2 - 4AC$
0465	30	2002	2100	2006	for conjugate
0466	36	2100	2003	2007	multuplication
0467	34	3000	2100	1607	$(B^2 - 4AC)(B^2 - 4AC)^* = r^2$
0470	34	3000	2100	1551	Compute r
0471	31	2000	2001	1434	Store r in TS_1
0472	34	3000	2100	1551	Compute $\sqrt{r}$
0473	31	1434	1475	2004	Load r
0474	31	2000	2001	1434	Store $\sqrt{r}$ in TS_1
0475	31	1440	1401	2000	$(B^2 - 4AC) = x$
0476	34	3000	2100	1507	$x/r = \cos\theta$
0477	31	2000	2001	1435	Store $\cos\theta$ in TS_2
0500	31	1331	1337	2002	Load 1
0501	34	3000	2100	1540	$1 + \cos\theta$
0502	36	2000	1331	2000	$(1 + \cos\theta)/2$
0503	34	3000	2100	1551	$\sqrt{(1 + \cos\theta)/2} = \cos\theta/2$
0504	31	1434	1475	2004	Load $\sqrt{r}$
0505	34	3000	2100	1500	$\sqrt{r}(\cos\theta/2) = \alpha'$
0506	31	2000	2001	1436	α' to TS_3
0507	31	1435	1476	2002	Load $\cos\theta$
0510	31	1331	1337	2000	Load 1
0511	34	3000	2100	1516	$1 - \cos\theta$
0512	36	2000	1331	2000	$(1 - \cos\theta)/2$
0513	34	3000	2100	1551	$\sqrt{(1 - \cos\theta)/2} = \sin\theta/2$
0514	31	1434	1475	2004	Load $\sqrt{r}$
0515	34	3000	2100	1500	$\sqrt{r}(\sin\theta/2) = \beta'$
0516	31	2000	2001	1434	β' to TS_1
0517	30	2000	2100	1451	Putaway β_0
0520	30	2000	2100	1453	and β_1 exps

LAGRANGIAN INTERPOLATION METHOD FOR HIGH DEGREE POLYNOMIALS

ADD	INST	M^1	M^2	M^3	REMARKS
0521	30	1436	2100	1450	Putaway α_0
0522	30	1436	2100	1452	and α_1 exps
0523	32	1475	1575	2107	Set β_0 mag positive
0524	30	2007	2100	1461	and
0525	36	2100	2007	1463	β_1 mag negative
0526	32	1477	1575	2107	Clear mag sign bit
0527	33	1403	2100	0533	If $\mathcal{L}_m(B^2 - 4AC) \geq 0$, 0533
0530	30	2007	2100	1462	No, putaway α_1 positive
0531	36	2100	2007	1460	and α_0 negative
0532	34	3000	2100	0535	Skip to 0535
0533	30	2007	2100	1460	Putaway α_0 positive
0534	36	2100	2007	1462	and α_1 negative
0535	05	3000	3000	1330	Load g_i
0536	31	1450	1460	2004	Load α_0
0537	31	1451	1461	2006	Load β_0
0540	34	3000	2100	1416	Subtract
0541	31	2000	2001	2004	Load for
0542	30	2002	2100	2006	conjugate
0543	36	2100	2003	2007	multiplication
0544	34	3000	2100	1607	Multiply and
0545	31	2000	2001	1434	store in TS_1
0546	05	3000	3000	1330	Load g_i
0547	31	1452	1462	2004	Load α_1
0550	31	1453	1463	2006	Load β_1
0551	34	3000	2100	1416	Subtract
0552	31	2000	2001	2004	Load for
0553	30	2002	2100	2006	conjugate
0554	36	2100	2003	2007	multiplication
0555	34	3000	2100	1607	Multiply and
0556	31	2000	2001	1436	store in TS_3
0557	05	3000	3000	1330	Load g_i
0560	33	1434	1436	0566	If $TS_1 \geq TS_3$, 0566
0561	33	1436	1434	0563	If $TS_3 \geq TS_1$, 0563
0562	34	1475	1477	0566	If $TS_1 \geq TS_3$, 0566
0563	31	1451	1461	2006	Load α_0
0564	31	1450	1460	2004	and β_0
0565	34	3000	2100	0570	or
0566	31	1453	1463	2006	load α_1
0567	31	1452	1462	2004	and β_1
0570	34	3000	2100	1410	$B \pm \sqrt{B^2 - 4AC}$
0571	04	3000	3000	1330	Store in B
0572	31	1307	1350	2000	Load
0573	31	1327	1370	2002	$f(z_i)$
0574	35	1304	1331	2004	Load
0575	36	2100	1345	2005	$-2\Delta_i$
0576	35	1324	1331	2006	for
0577	36	2100	1365	2007	$-2\Delta_i \cdot f(z_i)$

LAGRANGIAN INTERPOLATION METHOD FOR HIGH DEGREE POLYNOMIALS

ADD	INST	M^1	M^2	M^3	REMARKS
0600	34	3000	2100	1607	Multiply
0601	31	1330	1371	2004	Load
0602	31	1332	1373	2006	$B \pm \sqrt{B^2 - 4AC}$
0603	34	3000	2100	1650	Divide for λ_{i+1}
0604	04	3000	3000	1330	Store in B
0605	31	1302	1343	2000	Load
0606	31	1322	1363	2002	z_i
0607	31	1301	1342	2004	Load
0610	31	1321	1362	2006	z_{i-1}
0611	34	3000	2100	1416	$z_i - z_{i-1} = h_i$
0612	31	1330	1371	2004	Load
0613	31	1332	1373	2006	λ_{i+1}
0614	34	3000	2100	1607	$(\lambda_{i+1}) \cdot (h_i) = h_{i+1}$
0615	31	1302	1343	2004	Load
0616	31	1322	1363	2006	z_i
0617	34	3000	2100	1410	$z_i + h_{i+1} = z_{i+1}$
0620	04	3000	3000	1310	Store in A
0621	31	1302	1343	2004	Load
0622	31	1322	1363	2006	z_i
0623	34	3000	2100	1416	$z_{i+1} - z_i = \eta$
0624	31	1302	1343	2004	$\eta / z_i = \gamma$
0625	31	1322	1363	2006	
0626	34	3000	2100	1650	
0627	33	1454	2000	0631	If $\epsilon > \eta$ for real part, 0631
0630	34	3000	2100	0632	No, evaluate $f(z_{i+1})$
0631	33	1454	2002	0725	If $\epsilon > -\eta$ for imag part, 0725
0632	26	1372	1466	2007	Build and
0633	35	1470	2007	0656	store testword
0634	30	1470	2100	2007	Reset A_k pickup
0635	35	2007	0264	0646	in 0646
0636	31	1310	1351	2004	Load
0637	31	1312	1353	2006	z_{i+1}
0640	31	0100	0101	2000	Load
0641	31	1333	1337	2002	A_n
0642	32	0657	1636	1635	Set exit in cpx. mpy. routine
0643	34	3000	2100	1611	Transfer into cpx. mpy.
0644	31	2002	2003	1602	Temp. store imag part
0645	32	0677	1636	1550	Set exit in add routine
0646	31	0104	0105	2002	Load A_k
0647	34	3000	2100	1542	Transfer into add routine
0650	31	1602	1643	2002	Load imag part for cpx. mpy.
0651	31	1310	1351	2004	Reload z_{i+1}
0652	31	1312	1353	2006	for next term
0653	35	0646	1466	0646	Modify A_k pickup
0654	34	0646	0656	0660	If A_o not taken,
0655	34	3000	2100	1611	repeat back
0656	31	0104	0105	2000	Testword storage
0657	00	0000	0000	0644	Constant for 0642
0660	04	3000	3000	1440	Store $f(z_{i+1})$

LAGRANGIAN INTERPOLATION METHOD FOR HIGH DEGREE POLYNOMIALS

ADD	INST	M^1	M^2	M^3	REMARKS
0661	31	1307	1350	2004	Load
0662	31	1327	1370	2006	$f(z_i)$
0663	34	3000	2100	1650	$f(z_{i+1})/f(z_i)$
0664	31	2000	2001	2004	Load quotient
0665	30	2002	2100	2006	for complex
0666	36	2100	2003	2007	conjugate
0667	34	3000	2100	1607	multiplication
0670	33	0676	2000	0700	If
0671	33	2000	0676	0673	$\lvert f(z_{i+1})/f(z_i)\rvert^2$
0672	34	0677	2001	0700	is greater than
0673	36	1330	1637	1330	100; halve
0674	36	1332	1637	1332	λ_{i+1} and
0675	34	3000	2100	0605	recompute z_{i+1}
0676	00	0000	0000	0007	Floating point
0677	00	5200	0000	0650	100
0700	17	2010	3000	0702	If SEN 1 up, print iteration
0701	34	3000	2100	0706	No, skip print
0702	21	1310	2100	0001	Print
0703	21	1351	2100	0001	z_{i+1}
0704	21	1312	2100	0001	real and
0705	21	1353	2100	0001	imaginary
0706	31	1301	1342	1300	Shift z_{i-1}
0707	31	1321	1362	1320	to z_{i-2}
0710	31	1302	1343	1301	Shift z_i
0711	31	1322	1363	1321	to z_{i-1}
0712	31	1310	1351	1302	Shift z_{i+1}
0713	31	1312	1353	1322	to z_i
0714	31	1330	1371	1303	Shift λ_{i+1}
0715	31	1332	1373	1323	to λ_i
0716	31	1306	1347	1305	Shift $f(z_{i-1})$
0717	31	1326	1367	1325	to $f(z_{i-2})$
0720	31	1307	1350	1306	Shift $f(z_i)$
0721	31	1327	1370	1326	to $f(z_{i-1})$
0722	31	1440	1401	1307	Shift $f(z_{i+1})$
0723	31	1442	1403	1327	to $f(z_i)$
0724	34	3000	2100	0361	Reiterate
0725	33	1312	1455	0751	If imag portion significant, 0751
0726	35	0731	1466	0735	Initialize 0735
0727	32	0750	1677	0737	Initialize 0737
0730	32	0747	1640	0740	Initialize 0740
0731	31	0100	0101	2000	Load A_n
0732	25	2001	1351	2003	} $A_n z_i$
0733	35	2000	1310	2002	
0734	31	2002	2003	2002	} $A_{n-1} + A_n z_i$
0735	31	0104	0105	2000	
0736	34	3000	2100	1540	
0737	30	2000	2100	0104	Put away
0740	30	2001	2100	0105	terms of suppressed equation
0741	35	0735	1466	0735	Modify to suppress

LAGRANGIAN INTERPOLATION METHOD FOR HIGH DEGREE POLYNOMIALS

ADD	INST	M^1	M^2	M^3	REMARKS
0742	35	0737	0264	0737	eqution to next
0743	35	0740	0264	0740	lower degree
0744	34	0656	0735	0732	If extraction not complete, 0732
0745	36	1372	1331	1372	Reduce N' by 1
0746	34	3000	2100	1013	Transfer to conversion routine
0747	00	0000	0000	0103	Constant for 0730
0750	00	0000	0000	0102	Constant for 0727
0751	35	1310	1331	1303	Build 2a term
0752	30	1351	2100	1344	in quadratic factor
0753	05	3000	3000	1310	Build
0754	31	2002	2003	2006	$(a^2 + b^2)$
0755	30	2000	2100	2004	term in
0756	36	2100	2001	2005	quadratic
0757	34	3000	2100	1607	factor
0760	31	2000	2001	1304	
0761	35	0766	1473	2007	Initialize
0762	35	2007	0264	0770	0770
0763	32	1437	1677	1002	Initialize 1002
0764	35	1002	1331	2007	Initialize
0765	32	2007	1535	1003	1003
0766	31	0100	0101	1300	A_n to TS_a
0767	31	0102	0103	1301	A_{n-1} to TS_b
0770	31	0114	0115	1302	A_{n-2} to TS_c
0771	35	1300	1303	2000	}
0772	25	1341	1344	2001	} A_n x a to 2004, 2005
0773	31	2000	2001	2004	}
0774	35	1300	1304	2000	}
0775	25	1341	1345	2001	} A_n x $(a^2 + b^2)$ to 2006, 2007
0776	31	2000	2001	2006	}
0777	31	1301	1342	2000	Load
1000	31	1302	1343	2002	A_{n-1}
1001	34	3000	2100	1410	Add
1002	30	1300	2100	0110	Putaway A_n of
1003	30	1341	2100	0111	suppressed equation
1004	31	2000	2001	1300	Set up for
1005	31	2002	2003	1301	next term
1006	35	0770	1466	0770	Modify to pick up A_{n-3}
1007	35	1002	0264	1002	Modify putaway
1010	35	1003	0264	1003	Modify putaway
1011	34	0656	0770	0770	Has extraction been completed?
1012	36	1372	0264	1372	Yes, reduce N' by 2
1013	30	1351	1310	1301	Form fractional part (1)
1014	36	1310	0261	2000	Build shift control word
1015	30	1351	2000	2004	Form integer part
1016	35	2100	2100	1302	Clear sign storage
1017	32	1351	1166	1302	Store sign
1020	34	2004	2100	1102	If integer ≥ 0, 1102
1021	34	3000	2100	1031	No, 1031

(1) The remainder of Channel 10 and Channel 11 is a binary floating point to decimal conversion which would be part of the overall program.

LAGRANGIAN INTERPOLATION METHOD FOR HIGH DEGREE POLYNOMIALS

ADD	INST	M^1	M^2	M^3	REMARKS
1022	12	0000	0000	0000	Suppress sign bits
1023	10	0100	0000	0000	Print control
1024	00	2400	0000	0000	Alphabetic print
1025	00	1515	2700	0000	Alphabetic print
1026	10	0000	0100	0000	Print control
1027	00	3115	1515	1500	Alphabetic print
1030	30	2005	2100	1300	Reset command
1031	30	1301	2100	2000	Load fraction
1032	30	1167	2100	2007	Load testword
1033	26	2000	0257	2000	Multiply for most sig. digit
1034	27	2007	0256	2007	Shift testword
1035	32	2001	0254	2007	Extract digit into TW storage
1036	37	2007	2100	1033	Repeat loop if o'flo
1037	30	2007	2100	1301	Putaway converted fraction
1040	32	1016	1022	1301	Suppress sign
1041	34	1163	1067	1043	Print editing
1042	34	3000	2100	1170	commands
1043	34	1331	2004	1057	Print editing
1044	27	1101	2100	2007	commands
1045	32	1300	0260	2100	Extract most sig. dec. digit
1046	34	2000	2100	1053	If 2000 $\geq$ 0, 1053
1047	21	1024	1023	0001	If not print space
1050	30	1300	0256	1300	Shift word 4 left
1051	30	2007	0256	2007	and change print control
1052	34	3000	2100	1045	Return to test next digit
1053	32	1302	1022	1300	Enter sign
1054	21	1300	2007	0001	Print integer
1055	21	1025	1026	0001	Print decimal point
1056	34	3000	2100	1066	Skip to 1066
1057	21	1027	1140	0001	Print editing command
1060	32	1302	1166	2100	Build sign print
1061	27	2000	0262	2000	when only fractional
1062	34	2000	2100	1065	part involved
1063	21	1157	1026	0001	Print +.
1064	34	3000	2100	1066	Skip to 1066
1065	21	1160	1026	0001	Print -.
1066	21	1301	1101	0001	Print fractional part
1067	33	1455	1312	1074	If imag part insig, 1074
1070	21	1161	1165	0002	No, print $\pm$ j
1071	35	1163	2100	1067	Set skip command in 1067
1072	31	1312	1353	1310	Set up convert
1073	34	3000	2100	1013	imaginary part
1074	21	1154	1023	0001	Print carriage return
1075	34	1372	2100	0310	If N' $\geq$ 0, next root
1076	22	0000	0000	0000	If N' = 0, halt
1077	35	1164	2100	1067	Restore 1067
1100	34	3000	2100	1074	Return to "test for end"
1101	01	0000	0000	0001	Constant for 1044

LAGRANGIAN INTERPOLATION METHOD FOR HIGH DEGREE POLYNOMIALS

ADD	INST	M^1	M^2	M^3	REMARKS
1102	35	2100	2100	2005	High
1103	31	1113	2004	2000	integer
1104	32	2000	1114	2100	conversion
1105	30	2000	1115	2000	routine
1106	35	1116	2000	1107	
1107	35	1126	2100	1110	
1110	23	2004	1140	2000	
1111	26	2000	1147	2000	
1112	34	3000	2100	1142	
1113	02	0000	0000	0002	
1114	00	0000	0000	0174	
1115	00	0000	0000	0026	
1116	35	1115	2100	1110	
1117	23	2004	1127	2000	
1120	23	2004	1130	2000	
1121	23	2004	1131	2000	
1122	23	2004	1132	2000	
1123	23	2004	1133	2000	
1124	23	2004	1134	2000	
1125	23	2004	1136	2000	
1126	23	2004	1137	2000	
1127	00	0167	1531	1777	
1130	00	0013	3274	0777	
1131	00	0001	1422	2377	
1132	00	0000	0750	2177	
1133	00	0000	0060	6477	
1134	00	0000	0004	7037	
1135	34	3000	2100	1151	
1136	00	0000	0000	0307	
1137	00	0000	0000	0023	
1140	10	0000	0000	0100	
1141	26	2000	1155	2000	
1142	30	2005	1150	2005	
1143	35	1110	1156	1110	
1144	35	2001	2005	2005	
1145	33	1110	1153	1141	
1146	34	3000	2100	1030	
1147	00	0000	0000	0024	
1150	00	0000	0000	0004	
1151	35	2100	2100	2005	
1152	34	3000	2100	1146	
1153	23	2004	1140	2000	
1154	00	3400	0000	0000	
1155	00	0000	0000	0012	
1156	00	0000	0001	0000	

ADD	INST	M^1	M^2	M^3	REMARKS
1157	00	2352	2764	6400	Print configuration for 1063
1160	00	2252	2764	6400	Print configuration for 1065
1161	00	2315	3621	3264	Print configuration for 1070
1162	00	4264	6464	3232	Print configuration for 1070
1163	34	3000	2100	1077	Dummy reset for 1071
1164	33	1455	1312	1074	Dummy reset for 1074
1165	10	0000	0000	0000	Print control - alphabetic
1166	02	0000	0000	0000	Sign extractor
1167	00	0421	0421	0420	Testword for 1072
1170	32	1016	1022	1302	Suppress integer sign when
1171	34	2004	2100	1044	printing imaginary portion
1172	21	1174	1175	0001	or print 0.
1173	34	3000	2100	1066	and return to 1066
1174	00	5227	0000	0000	Print configuration for 1172
1175	10	0001	0000	0000	Print control word
1176 - 1177					Not required

Channel 12 is used for a memory check-sum routine in the root-solving routine and is available for other use when this program is incorporated in the overall synthesis program.

Channel 13 is used for temporary storage and for a few constants which are listed below. NOTE: This program is coded for minimum access only, i.e., adjacent cells in the main memory are numbered 00, 41, 02, 43, 04, 45, etc.

1331, 1337 - floating point "1", must be addressed individually
1333, 1335 - floating point "0", must be addressed individually

1372 - N' - the degree of the reduced equation

1374	00	3000	3000	0200	Testword for 0305
1376	00	0000	0000	0010	Increment for 0303, 0304

Channels 14, 15, and 16 contain complex floating point arithmetic routines, i.e.,

Complex Addition	1410
Complex Subtraction	1416
Simple Multiplication	1500
Simple Division	1507
Simple Subtraction	1516
Simple Addition	1540
Square Root	1551
Complex Multiplication	1607
Complex Division	1650

LAGRANGIAN INTERPOLATION METHOD FOR HIGH DEGREE POLYNOMIALS

ADD	INST	M^1	M^2	M^3	REMARKS
1400 - 1407	Used for temporary storage of operands				
1410	30	3000	1505	1400	Prepare
1411	32	1400	1640	1433	exit
1412	04	3000	3000	1400	Store operands
1413	32	1472	1535	1424	Set 1424
1414	32	1471	1677	1430	and 1430 for addition
1415	34	3000	2100	1423	Transfer to 1423
1416	30	3000	1603	1536	Prepare
1417	32	1536	1515	1433	exit
1420	04	3000	3000	1400	Store operands
1421	32	1465	1636	1424	Set 1424
1422	32	1467	1636	1430	and 1430 for subtraction
1423	31	2004	2005	2002	Reposition real operands
1424	34	3000	2100	1516	Transfer for ± reals
1425	31	2000	2001	1400	Store sum of reals
1426	31	1402	1443	2000	Load imaginary
1427	31	1406	1447	2002	operands
1430	34	3000	2100	1516	Transfer for ± imaginaries
1431	31	2000	2001	2002	Reposition imaginary sum
1432	31	1400	1441	2000	Load real sum
1433	34	3000	2100	0624	Exit
1434 - 1436	Temporary storage				
1437	00	0000	0000	0100	Decrementer for 0306, 0307
1440 - 1453	Temporary storage				
1454	02	0000	0000	0044	
1455 - 1463	Temporary storage				
1464	00	0000	7777	7777	M^2, M^3 extractor for 0333
1465	00	0000	0000	1516	Constant for 1421
1466	00	0002	0002	0000	Constant for 0323
1467	00	0000	0000	1516	Constant for 1422
1470	31	0102	0103	2000	Dummy reset for 0326
1471	00	0000	0000	1540	Constant for 1414
1472	00	0000	0000	1540	Constant for 1413
1473	00	0004	0004	0000	Constant for 0332, 0336
1474	00	0002	0000	0000	Incrementer (various)
1475 - 1477	Temporary storage				
1500	30	3000	1505	2006	Prepare
1501	32	2006	1677	1550	exit
1502	35	2000	2004	2000	Add exponents
1503	25	2001	2005	2001	Multiply magnitudes
1504	34	3000	2100	1547	Transfer to exit
1505	02	0000	0000	0030	Shift control for 1500
1506	00	0000	0000	3777	M^3 extractor
1507	30	3000	1676	2006	Prepare
1510	32	2006	1677	1550	exit
1511	36	2000	2004	2000	Subtract exponents
1512	23	2001	2005	2001	Divide magnitudes
1513	37	2001	3000	1530	If overflow, 1530
1514	34	3000	2100	1547	Transfer to exot
1515	00	0000	0000	7777	M^3 extractor

LAGRANGIAN INTERPOLATION METHOD FOR HIGH DEGREE POLYNOMIALS

ADD	INST	M^1	M^2	M^3	REMARKS
1516	30	3000	1505	2006	Prepare
1517	32	2006	1677	1550	exit
1520	36	2100	2003	2003	Change sign 2nd operand
1521	34	3000	2100	1542	Transfer into add
1522	36	2000	2002	2006	Form shift control
1523	30	2001	2006	2007	Shift 1st operand
1524	30	2002	2100	2000	Shift 2nd exponent to Ans. cell
1525	35	2003	2007	2001	Add operands
1526	37	2001	3000	1530	If overflow, 1530
1527	34	3000	2100	1547	Transfer to exit
1530	27	2001	1644	2001	Normal
1531	30	2001	1644	2001	correct
1532	27	2001	1646	2001	overflow
1533	35	2000	1646	2000	procedure
1534	34	3000	2100	1547	Transfer to exit
1535	00	0000	0000	7777	M^3 extractor
1536	00	0000	0000	0624	Prepare exit storage
1537	02	0000	0000	0001	Shift control for 1555
1540	30	3000	1505	2006	Prepare
1541	32	2006	1515	1550	exit
1542	33	2002	2000	1522	If 2002 $\geq$ 2000, 1522
1543	36	2002	2000	2006	No, form shift control
1544	30	2003	2006	2007	Shift 2nd operand
1545	35	2001	2007	2001	Add operands
1546	37	2001	3000	1530	If overflow, 1530
1547	31	2000	2001	2000	Normalize
1550	34	3000	2100	....	Exit
1551	30	3000	1676	2006	Prepare exit
1552	32	2006	1535	1550	from square root routine
1553	35	2000	0261	2000	Multiply by 2^{44}
1554	32	2000	1331	2104	Extract last bit of exp into 2004
1555	30	2000	1644	2006	Take square root of exp
1556	34	2004	2100	1561	If exp odd, magnitude has been divided by 2, by 1555
1557	30	2001	1644	2007	If exp even, divide magnitude by 2
1560	34	3000	2100	1562	Skip to iteration
1561	30	2001	2100	2007	Load magnitude for iteration
1562	30	1575	2100	2002	Load a_i
1563	23	2007	2002	2001	$(x/2)/a_i$
1564	36	2001	2002	2001	$(x/2a_i) - a_i$
1565	30	2001	1644	2001	$1/2(x/2a_i) - a_i = a_{i+1} - a_i$
1566	34	2001	2100	1570	Has process converged?
1567	34	3000	2100	1572	Yes, 1572; no, 1570
1570	35	2002	2001	2002	a_{i+1} replaces a_i
1571	34	3000	2100	1563	and reiterate
1572	25	2002	1577	2001	Multilpy by $\sqrt{2}/2$
1573	36	2006	1576	2000	Divide by 2^{21}
1574	34	3000	2100	1547	Transfer to exit

LAGRANGIAN INTERPOLATION METHOD FOR HIGH DEGREE POLYNOMIALS

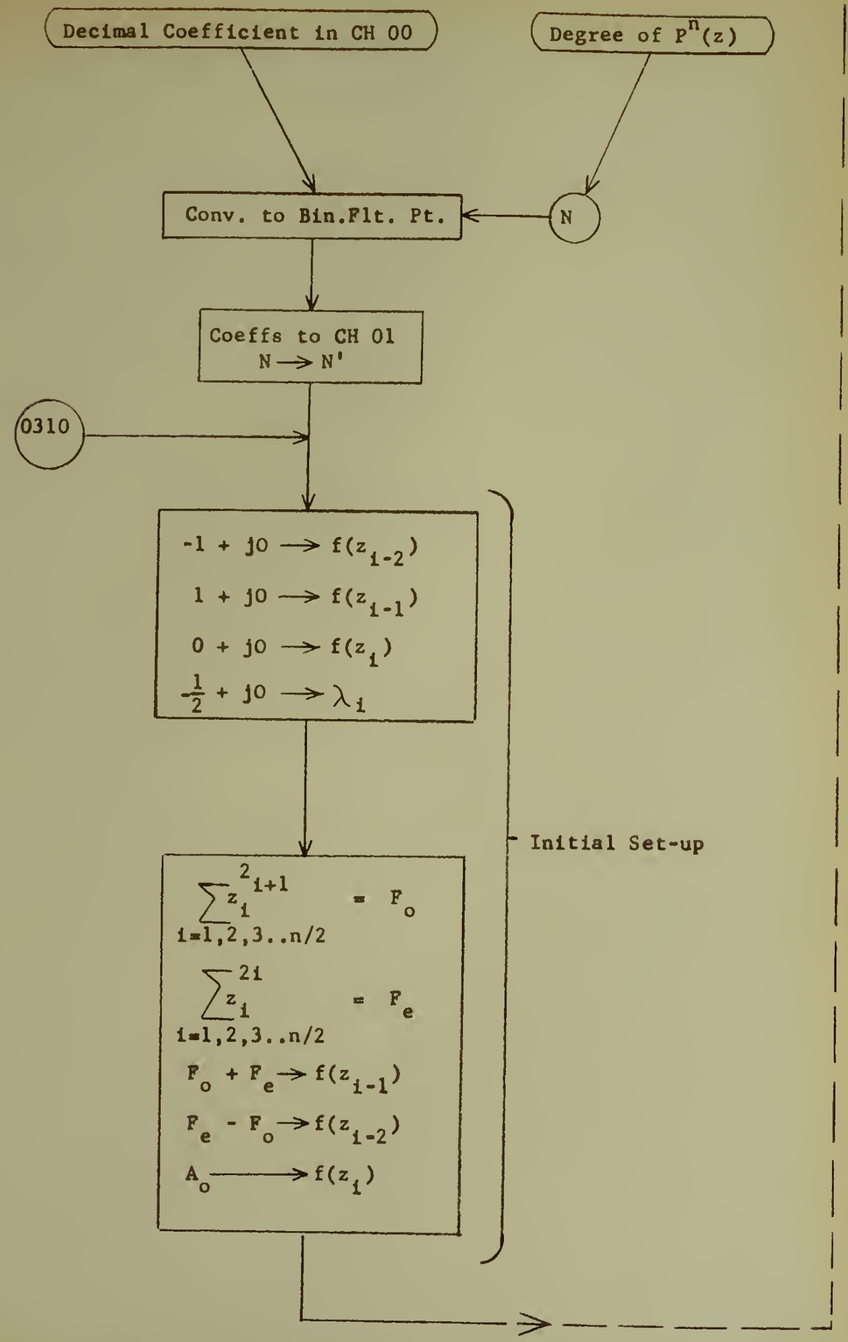
ADD	INST	M^1	M^2	M^3	REMARKS
1575	00	7777	7777	7777	Constant for 1562
1576	00	0000	0000	0021	Constant for 1573
1577	00	5520	2363	1500	Constant for 1572
1600 - 1602	Used for temporary storage of operands				
1603	02	0000	0000	0030	Shift control for 1607
1604 - 1606	Used for temporary storage of operands				
1607	30	3000	1603	1601	Prepare
1610	32	1601	1636	1635	exit
1611	04	3000	3000	1600	Store operands
1612	34	3000	2100	1500	Multiply reals
1613	30	2000	2100	1601	and store results
1614	30	2001	2100	1605	in 1601, 1605
1615	35	1602	1606	2000	Multiply imaginaries
1616	25	1643	1647	2001	to obtain negative real
1617	31	2000	2001	2002	and shift for subtraction
1620	31	1601	1605	2000	from previous real product
1621	34	3000	2100	1516	Transfer to subtract
1622	30	2000	2100	1601	Putaway real product
1623	30	2001	2100	1605	in 1601, 1605
1624	35	1602	1604	2000	Multiply for
1625	25	1643	1645	2001	first cross-product term
1626	31	2000	2001	2002	and hold in 2002, 2003
1627	35	1600	1606	2000	Multiply second
1630	25	1641	1647	2001	cross-product term
1631	31	2000	2001	2000	and prepare for addition
1632	34	3000	2100	1540	Add for imaginary cross-products
1633	31	2000	2001	2002	Put results in2002, 2003
1634	31	1601	1605	2000	Insert real product
1635	34	3000	2100	0627	Exit
1636	00	0000	0000	7777	M^3 extractor for 1610
1637	00	0000	0000	0001	Shifter for 1656
1640	00	0000	0000	7777	M^3 extractor
1641	Used for temporary storage				
1642	00	0000	0000	0001	Shift control word
1643	Used for temporary storage				
1644	02	0000	0000	0001	Shift control word
1645	Used for temporary storage				
1646	00	0000	0000	0001	Shift Control word
1647	Used for temporary storage				
1650	30	3000	1676	1601	Prepare $\frac{(A+jB)}{(C+jD)}$
1651	32	1601	1677	1635	exit
1652	04	3000	3000	1600	Store operands
1653	30	2004	1637	2000	Form
1654	25	2005	1645	2001	C^2
1655	31	2000	2001	2000	and normalize

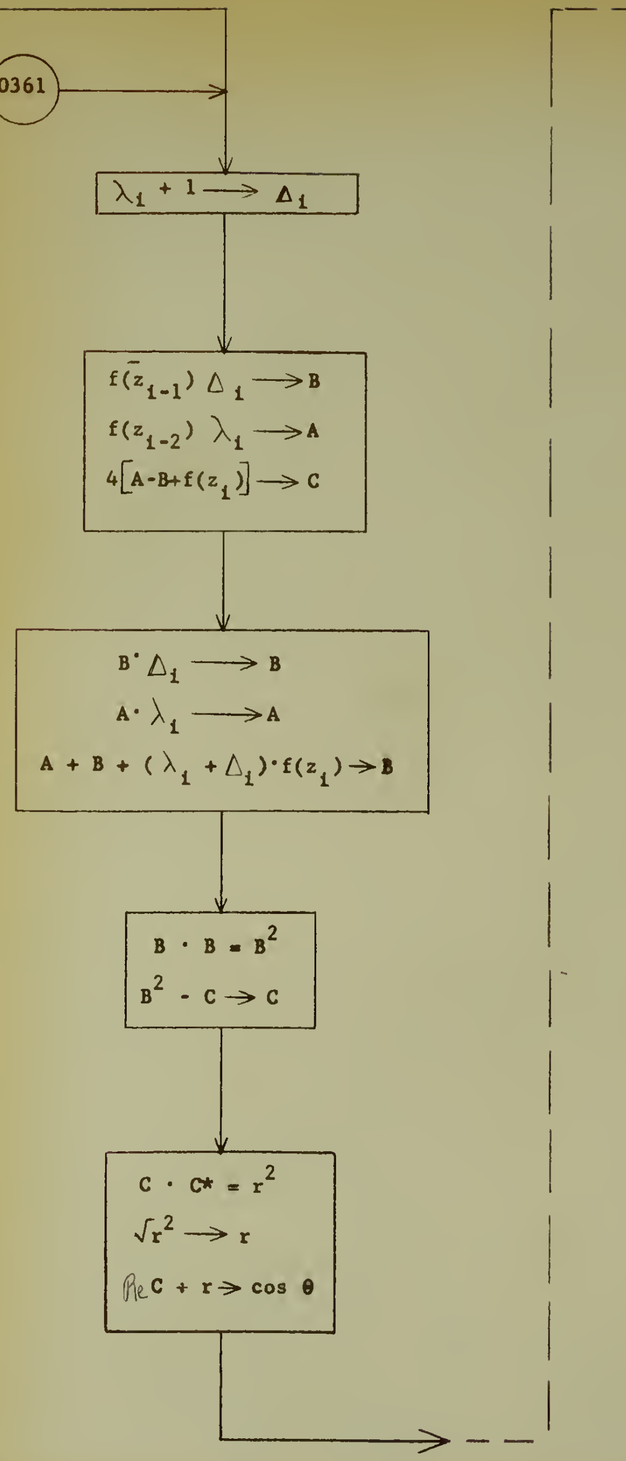
LAGRANGIAN INTERPOLATION METHOD FOR HIGH DEGREE POLYNOMIALS

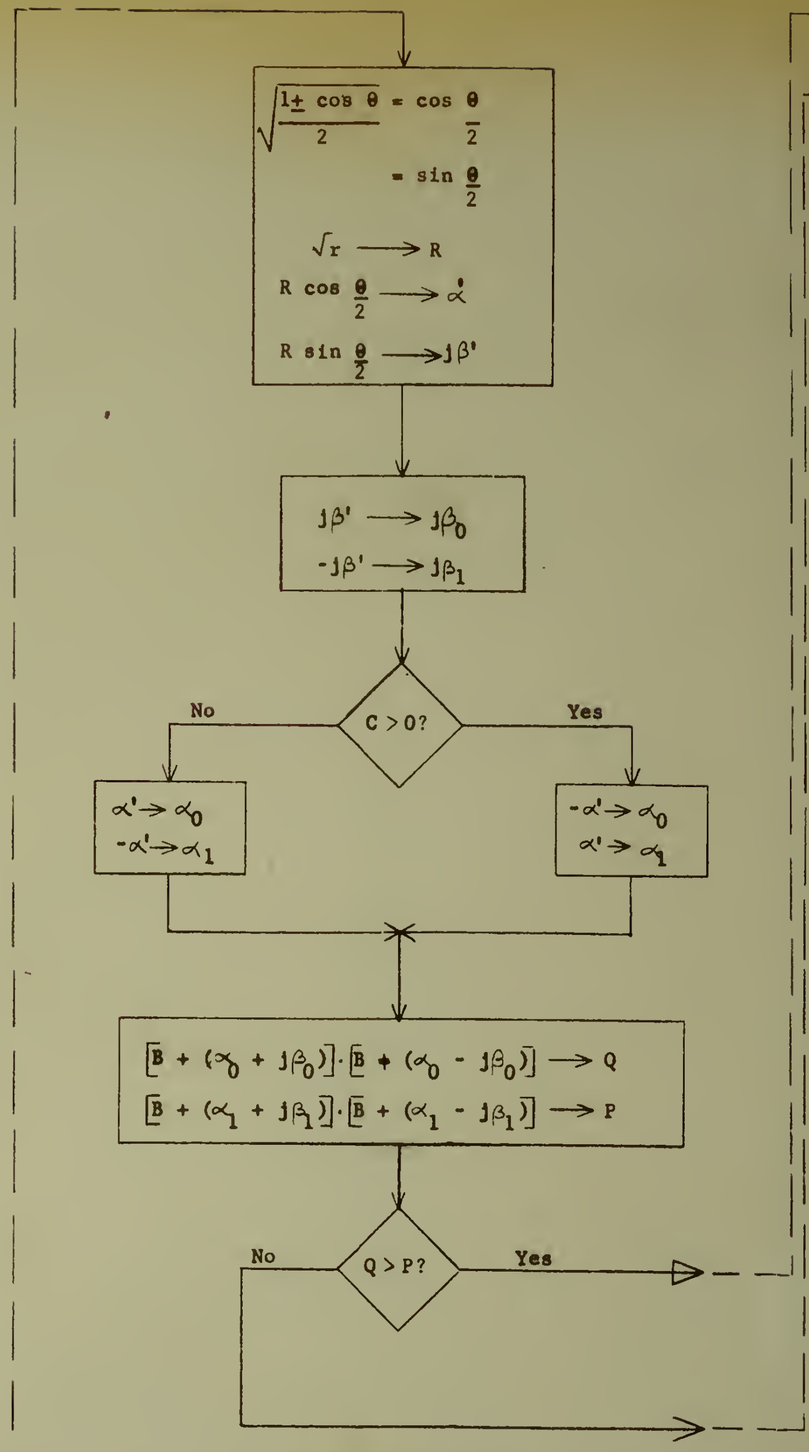
ADD	INST	M^1	M^2	M^3	REMARKS
1656	30	2006	1637	2002	Form
1657	25	2007	1647	2003	D^2
1660	31	2002	2003	2002	and normalize
1661	34	3000	2100	1540	$(C^2 + D^2)$
1662	31	2000	2001	2004	Shift to 2004, 2005
1663	31	1604	1645	2000	Load C
1664	34	3000	2100	1507	$C/ (C^2 + D^2)$
1665	30	2000	2100	1604	Store in
1666	30	2001	2100	1645	1604, 1645
1667	31	1606	1647	2000	Load D
1670	34	3000	2100	1507	$D/ (C^2 + D^2)$
1671	30	2000	2100	1606	Store in 1606,
1672	36	2100	2001	1647	1647 with negative sign
1673	31	1604	1645	2004	Load conjugate
1674	31	1600	1641	2000	quotient and
1675	34	3000	2100	1612	transfer to multiply
1676	02	0000	0000	0030	Shift control word
1677	00	0000	0000	7777	M^3 extractor

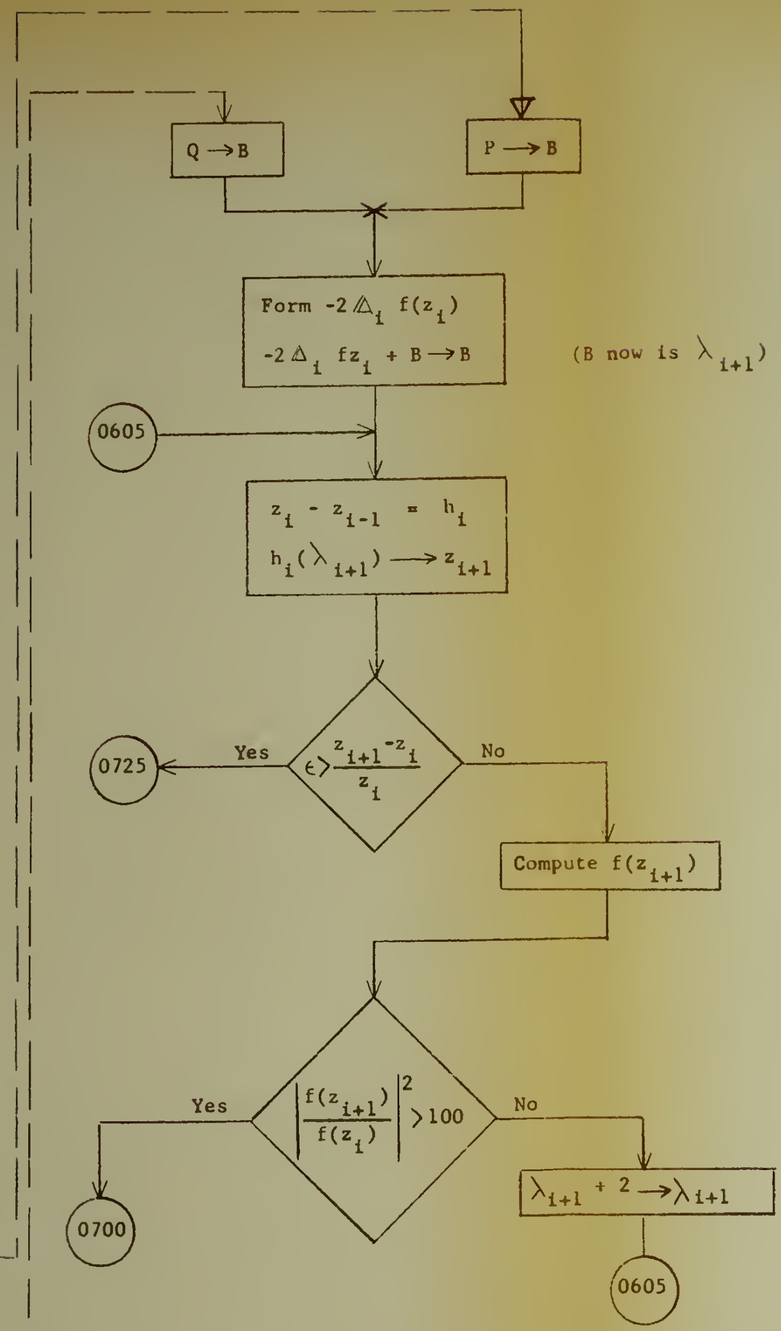
FINIS

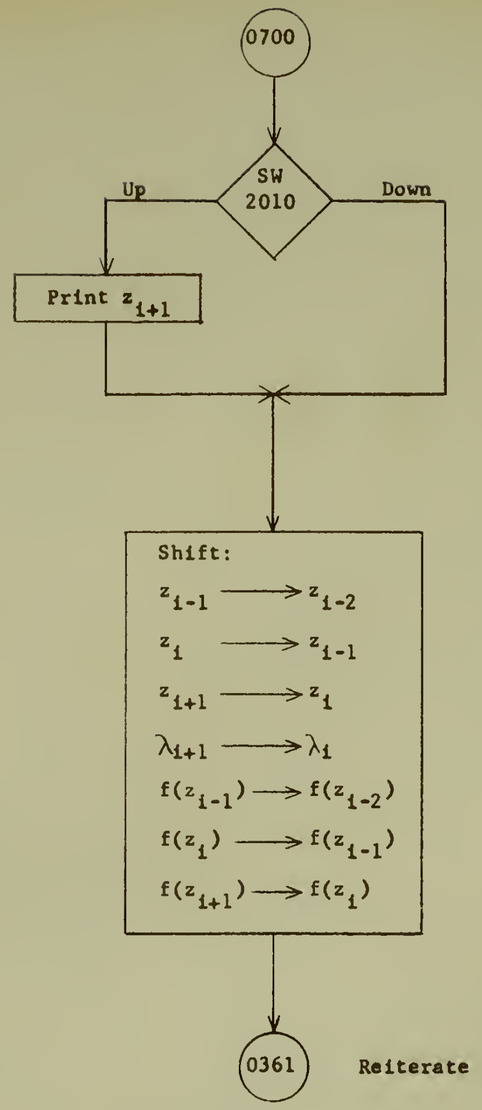
FLOW CHART OF LAGRANGIAN INTERPOLATION METHOD
FOR SOLUTION OF ALGEBRAIC EQUATIONS OF HIGH DEGREE

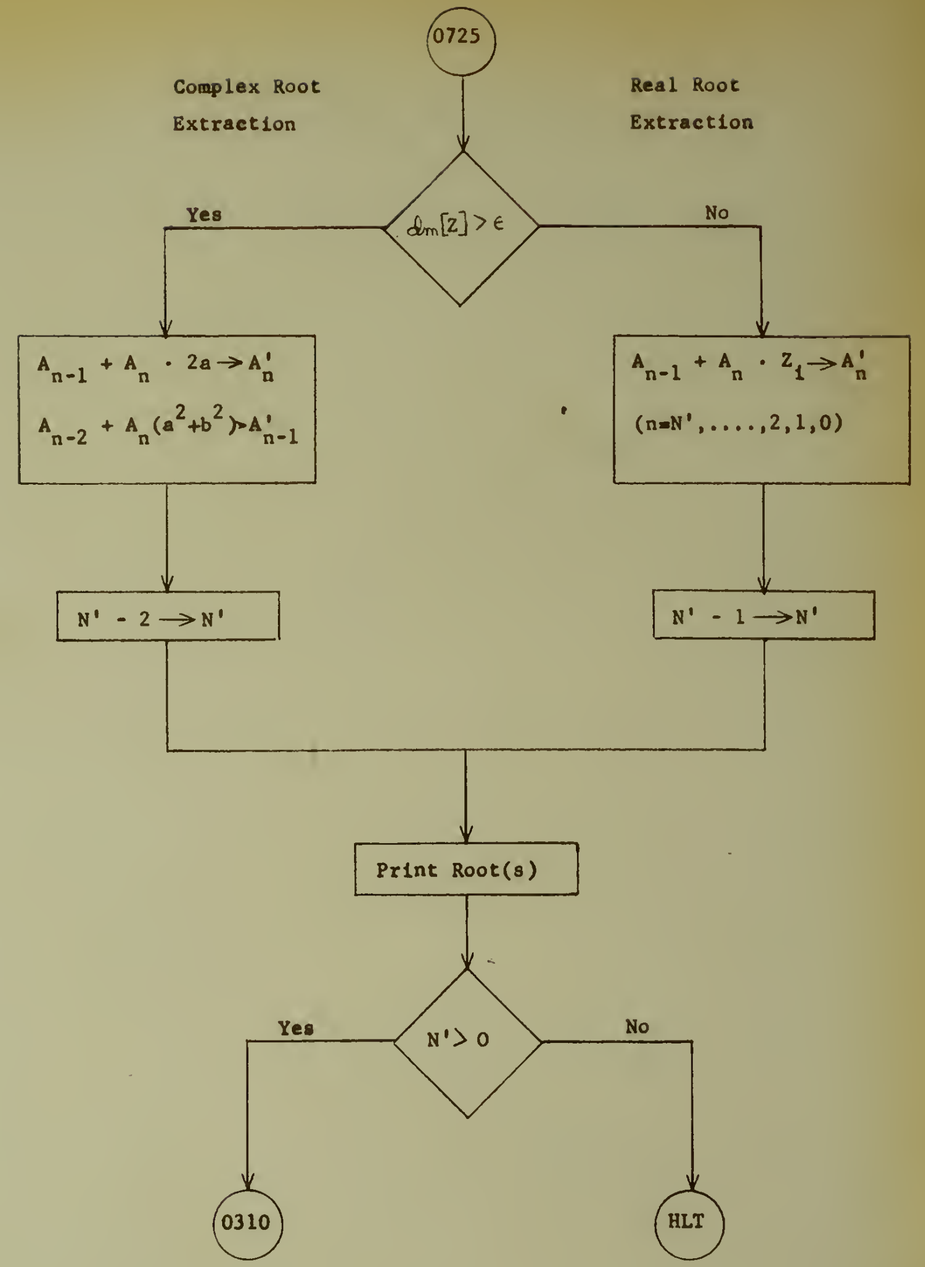







Q → B
P → B
Form $-2\Delta_i f(z_i)$
$-2\Delta_i fz_i + B \rightarrow B$
(B now is λ_{i+1})
0605
$z_i - z_{i-1} = h_i$
$h_i(\lambda_{i+1}) \rightarrow z_{i+1}$
$\epsilon > \frac{z_{i+1} - z_i}{z_i}$
Yes
No
0725
Compute $f(z_{i+1})$
$\left|\frac{f(z_{i+1})}{f(z_i)}\right|^2 > 100$
Yes
No
0700
$\lambda_{i+1} + 2 \rightarrow \lambda_{i+1}$
0605


0700
SW 2010
Up
Down
Print z_{i+1}
Shift:
$z_{i-1} \rightarrow z_{i-2}$
$z_i \rightarrow z_{i-1}$
$z_{i+1} \rightarrow z_i$
$\lambda_{i+1} \rightarrow \lambda_i$
$f(z_{i-1}) \rightarrow f(z_{i-2})$
$f(z_i) \rightarrow f(z_{i-1})$
$f(z_{i+1}) \rightarrow f(z_i)$
0361
Reiterate


0725
Complex Root Extraction
Real Root Extraction
$\mathcal{I}m[z] > \epsilon$
Yes
No
$A_{n-1} + A_n \cdot 2a \rightarrow A'_n$
$A_{n-2} + A_n(a^2+b^2) \rightarrow A'_{n-1}$
$A_{n-1} + A_n \cdot Z_1 \rightarrow A'_n$
$(n=N', \ldots, 2, 1, 0)$
$N' - 2 \rightarrow N'$
$N' - 1 \rightarrow N'$
Print Root(s)
$N' > 0$
Yes
No
0310
HLT

APPENDIX V

USE OF THE EXISTING PROGRAMS ON THE CRC-102A

All existing programs have been coded for minimum-access word channel and will not work in the sequential channels.

1. Dirichlet Power Series Expansion

Since the synthetic division and conversion program has not been written as yet, if $f_o(t)$ and $f_i(t)$ are given, $\{h\}_A$ must be computed by hand. If h(t) is given, it must be converted to $\{h\}_A$ by multiplying by Δt. As the program is now located, storage is available for 100 (octal) ordinates in floating point form if the decimal to floating point conversion in Channel 2 is to be used. If the conversion is accomplished prior to entry of data, 150 (octal) ordinates may be used.

$$S = 2\frac{T}{\Delta t} = 2NT$$

where S = total cells needed

T = number of time units problem is scaled over (note command 0423)

Δt = sampling interval

N = samples per time unit

The program may be relocated to accommodate more sampled ordinates.

The binary floating point numbers representing the ordinates of h(t) are entered in cells 0000 - 0117, exponents in even cells, magnitudes in odd; this allows 40 ordinates to be used.

The test switches determine the amount of interpolation performed. The output will be H*(s) in decimal form for as many terms requested. The number of terms is set by entering this datum as an octal number in J, cell 0540.

2. Matrix Inversion

The matrix inversion program takes the ascending power H*(s) terms in

packed floating point form (a conversion routine is available) in the word structure shown on page III-5 and computes the lossless system function, Z'(p). The terms are entered in Channel 0,0000, 0001, 0002, etc. Cells 0076 and 0077 are reserved for N, the order of the matrix which equals n, the number of poles desired or the degree of the denominator and n + r, the degree of the numerator respectively. For the example on page 21,

N = n = 4 (0076)

n + r = n - 2 = 2 (0077)

The commands 0300 through 0320 contain instructions which link the program to a packed binary floating point to decimal conversion routine located in 1571 - 1677.

If these commands are included, the program will yield the decimal coefficients of Z'p in the following form:

$$\frac{p_0 + p_1 s + p_2 s^2}{q_0 + q_1 s + q_2 s^2 + q_3 s^3 + q_4 s^4}$$

q_0 will be equal to 1, the results must be normalized to the following form for entry in the root-solving routine.

$$\frac{(M) \quad s^2 + p_1' s + p_0'}{s^4 + q_3' s^3 + q_2' s^2 + q_1' s + q_0'}$$

3. Lagrangian Interpolation Method

The procedure for the use of this program is clearly described in the description of sub-routines for the Computation Center (Routine 2.1).

It is recommended that the Channel 2 conversion routine of this program be replaced to provide the normalizing procedure required by 2. above. Channels 11 and 12 are available for the storage of the factored forms of the numerator and denominator.